MSX Machinetaalboek

MSXB4

DATA BECKER NEDERLANDS*

Dullin/Strassenburg

MSX Machine-taalboek

MSX Bibliotheek 4

DATA BECKER NEDERLANDS*

MSX Machine-taalboek

Dullin/Strassenburg

MSX Machine-taalboek

MSX Bibliotheek 4

DATA BECKER
NEDERLANDS*

Oorspronkelijke titel: Das Maschinensprachebuch zu MSX
© 1985 Data Becker GmbH, Düsseldorf
Vertaling: ProCread, Groningen
© 1985 Data Becker Nederlands
Omslagontwerp: M. van Keulen
Druk: Vonk Zeist
ISBN 90 449 3360 1
D/1986/0939/34
1e druk 1986
2e druk 1986

De keuze en de opbouw van de programma's die in dit boek voorkomen, zijn gebaseerd op hun educatieve waarde. Alle programma's zijn getest op hun juiste werking. De uitgever kan geen enkele verantwoordelijkheid voor eventueel optredende fouten aanvaarden.

Uit deze uitgave mag niets worden openbaar gemaakt en/of verveelvoudigd door middel van druk, fotokopie, microfilm of op welke andere wijze dan ook zonder voorafgaande schriftelijke toestemming van de uitgever.

Boeken en programma's van

DATA BECKER
NEDERLANDS*

worden in de handel gebracht door:
A.W. Bruna & Zoons Uitgeversmij b.v.,
Postbus 8411, 3503 RK Utrecht
en
A.W. Bruna & Zoon n.v.,
Antwerpsesteenweg 29a, 2630 Aartselaar.

INHOUDSOPGAVE

BIJLAGEN

Dankwoord

Een boek als dit schrijven is eigenlijk alleen leuk voor de auteurs. Bij medebewoners, vriendinnen en vrienden treden maar al te vaak frustraties op omdat de schrijvers zich moeten onderdompelen in "schrijversroes en computergekte" om een boek af te kunnen maken. We willen daarom iedereen bedanken die door de geboden hulp of het opgebrachte geduld heeft bijgedragen aan de totstandkoming van dit boek. In het bijzonder bedanken we Birgitt Mikutta, Kristin Grünewald en Andreas Bethmann voor hun onschatbare hulp bij de correctie van het manuscript. Ralph Dullin bedanken we voor het maken van de tekeningen en enkele tabellen.

Bovendien zijn we dank verschuldigd aan de firma Zilog Inc., USA, die de instructieset van de Z80 ter beschikking heeft gesteld.

Voorwoord

Toen we de veelbelovende MSX onder ogen kregen, waren we meteen enthousiast over deze computer. MSX-BASIC is werkelijk een uitstekende programmeertaal. Bij deze nieuwe computergeneratie is 'compatible' het toverwoord. Daarmee bedoelen we dat programma's die op een MSX computer draaien ook op alle andere machines met dezelfde standaard lopen. Deze bewering is niet uit de lucht gegrepen: we hebben deskundigen geraadpleegd en de programma's zelf uitvoerig getest.

Machinetaal is op enkele belangrijke punten superieur aan BASIC: programma's worden veel sneller uitgevoerd en nemen veel minder geheugenruimte in beslag. Dit boek beoogt de MSX-gebruiker vertrouwd te maken met machinetaal en daardoor te laten profiteren van deze eigenschappen. Machinetaal is niet de makkelijkste taal om te leren; wat moeten we bijvoorbeeld aan met een regel als deze:

```
3E,2A,21,00,00,01,C0,CD,15,08,C9
```

Ga nu niet gelijk bij de pakken neerzitten. Het wordt een stuk makkelijker als u de volgende wenken in acht neemt:

- bestudeer het boek hoofdstuk voor hoofdstuk
- probeer alle opgaven op te lossen
- lukt dat niet gelijk, werk dan het betreffende hoofdstuk nog eens rustig door

Genoeg goede raad: u kunt nu zelf met machinetaal aan de slag.

De schrijvers.

1 INLEIDING

1.1 Wat is machinetaal?

Machinetaal is de programmeertaal die de computer direct kan verwerken. De microprocessor (CPU: central processing unit) is het hart van de computer. Dit IC (integrated circuit) voert machinetaalinstructies uit, regelt de processen in de computer en stuurt de randapparatuur. Het type processor bepaalt de vorm van de machinetaal. De microprocessor van de MSX, de Z80, kent meer dan 600 instructies. Deze chip wordt in veel computers gebruikt. De meeste homecomputers zijn uitgerust met BASIC, een taal die makkelijk te leren is. Het MSX-BASIC munt uit door de hoeveelheid instructies. Een nadeel van BASIC is dat de CPU de instructies pas begrijpt als de BASIC-interpreter er machinetaal van heeft gemaakt. Dat kost tijd. Zodra de interpreter een BASIC-instructie signaleert, roept hij de machinetaalroutine op die de instructie uitvoert. Teken voor teken leest de interpreter een letter- of cijfercombinatie, afgebakend door spaties, dubbele punten, haakjes of komma's. Als zo'n combinatie niet overeenkomt met een BASIC-instructie in de tabel in het bedrijfssysteem, probeert de interpreter of het om een variabele gaat. Is dat ook niet het geval dan verschijnt er een foutmelding. Vindt de interpreter de instructie wel dan gaat hij naar het sprongadres, waar de volgende waarde wordt ingelezen en gecontroleerd (bij bijvoorbeeld MODE 2 is dat 2). Als de instructie is uitgevoerd, keert de computer naar de interpreter terug en begint het proces opnieuw. We besparen flink wat tijd door in machinetaal te programmeren. Bovendien neemt BASIC een hoop geheugenruimte in beslag en zijn de programmeermogelijkheden beperkt. Aan de andere kant heeft machinetaal het nadeel zeer abstract te zijn. Een programmeertaal is een compromis tussen mens en machine. Alle hogere programmeertalen, interpreters zowel als compilers, moeten worden vertaald voordat de computer ze kan uitvoeren. Een interpreter vertaalt de instructies stuk voor stuk en voert ze direct uit. Omdat eenzelfde instructie steeds opnieuw wordt vertaald, is een BASIC-programma zo gemakkelijk te veranderen. Een compiler daarentegen zet het gehele programma om in machinetaal, waarna de computer het uitvoert. Compileren neemt vrij veel tijd in beslag, maar daar staat tegenover dat de geproduceerde machinetaalcode sneller werkt. Een verandering in het programma vereist de compilatie van een hele nieuwe versie. In dit boek komt de assembler ter sprake, een compiler die assemblercode omzet in machinetaal. Zelf geschreven machinetaalprogramma's zijn sneller dan de machinetaalcodes die

een compiler produceert en zelfs tot 1000 keer sneller dan BASIC-programma's. De RETURN-instructie duurt in BASIC ongeveer 0.6 milliseconde, het machinetaalequivalent RET maar 2,5 microseconden. Machinetaal is in dit geval ongeveer 240 keer zo snel, bij POKE ongeveer 1000 keer. Deze verschillen zijn vooral interessant bij sorteer- en zoekopdrachten wanneer het gaat om grote aantallen gegevens en niet te vergeten in gevallen waarin de geheugeninhoud moet worden verschoven (bijvoorbeeld nodig bij scrolling en tekstprogramma's). Ook om Hi-Res grafiek te programmeren is BASIC te langzaam. BASIC is daarom niet geschikt voor spelletjes of business-grafiek. Machinetaalprogramma's zijn meestal korter dan BASIC-programmatuur; 500 bytes is al vrij lang. Dat scheelt aanmerkelijk in de geheugenruimte. Met machinetaal kunt u bijvoorbeeld I/O-bouwstenen programmeren en gegevens anders structureren dan in BASIC. Grote aantallen gegevens (denk aan tekstverwerking) kunnen op die manier beter in het geheugen worden ondergebracht. De assembler kent aan elke machinetaalcode (= een getal) een reeks symbolen toe:

- de afkorting van de instructie in het Engels, mnemonic genaamd
- operanden, die adressen, constanten en dergelijke aangeven

Een assemblerprogramma zet assemblertaal om in machinetaal. We noemen zo'n programma kortweg assembler. Dat is dus een compiler voor assemblertaal. Hexadecimale (zestientallige) getallen (machinetaal) gebruiken we alleen zo nu en dan als voorbeeld, in dit boek staat de programmering in assemblertaal voorop. De vertaling laten we aan de compiler (de assembler) over.

1.2 Het eerste machinetaalprogramma

Voer de volgende BASIC-regels in:

```
10 HL=0
20 A=42
30 VPOKE HL,A
40 HL=HL+1
50 IF HL<960 THEN 20
60 RETURN
```

Toets na deze regels in de directe modus SCREEN 0 in, vervolgens GOSUB 10 en kijk wat er gebeurt. Het volgende programma laadt het machinetaalprogramma dat dezelfde functie heeft:

```
10 CLEAR 200,&HEFFF
20 FOR I=&HF000 TO &HF00E
30 READ A
40 POKE I,A
50 NEXT I
60 DEF USR1=&HF000
70 END
80 DATA &H21,&H00,&H00,&H3E,42,&HCD,&HCD
90 DATA &H07,&H23,&H3E,04,&HBC,&H20,&HF5,&HC9
```

Toets weer in de directe modus SCREEN 0 in, RUN het programma en roep het op met X=USR1(1). De afwikkeling van het BASIC-programma duurt ongeveer 11 seconden, van het equivalent in machinetaal 0.1 seconde. Het BASIC-programma neemt 76 bytes in beslag, het machinetaalprogramma maar 15 (&HF000-&HF00E). Beide programma's nog even naast elkaar:

```
BASIC                          assembler

10 HL=0                        LD HL,0000
20 A=42                        LD A,42
30 VPOKE HL,A                  CALL &H7CD
40 HL=HL+1                     INC HL
50 IF HL<960 THEN 20           LD A,4
                               CP H
                               JR NZ,-11
60 RETURN                      RET
```

Verklaring:

Regel 10 de waarde voor variabele HL (register HL) wordt aan het begin van het beeldschermgeheugen gezet (LD staat voor LOAD (laden))

Regel 20 in A wordt de ASCII-code van het uit te voeren teken (*) opgeslagen

Regel 30 de waarde van A gaat naar het beeldschermgeheugen zodat het teken 1 keer wordt weergegeven

Experimenteer eens (in de directe modus) met waarden voor adres HL in het beeldschermgeheugen (kies HL tussen 0 en 1023) en voor A (de ASCII-code; kies hierbij waarden tussen 0 en 255), bijvoorbeeld POKE 10,65.

Regel 40 verhoogt variabele HL of register HL met 1 (INC staat voor INCREASE (verhogen))

Regel 50 controle of HL groter is dan 1023, dus of het einde van het beeldscherm is bereikt; in machinetaal neemt dit twee instructies in beslag: CP (COMPARE (vergelijken)), JR (JUMP RELATIVE (conditionele sprong)), NZ (NON ZERO (niet nul))

Regel 60 RET (RETURN) beëindigt de subroutine.

De assemblerlisting van het machinetaalprogramma:

adres	machinetaal	regel-nr. in BASIC-prog.	assembler-taal	commentaar
F000	210000	10	LD HL,0	start van beeldschermgeheugen
F003	3E2A	20	LD A,&H2A	42 (&H2A), de ASCII-code van *
F005	CDCD07	30	CALL &H7CD	komt overeen met de VPOKE-routine
F008	23	40	INC HL	HL=HL+1
F009	3E04	50	LD A,4	
F00B	BC	60	CP H	H>4?
F00C	20F5	70	JR NZ,&HF003	nee, dan nog eens
F00E	C9	80	RET	terug naar BASIC

In de volgende paragrafen komt machinetaal systematisch aan de orde en worden de voorbeelden uit deze paragraaf verklaard.

1.3 Talstelsels

In de vorige paragraaf was &H het teken voor een hexadecimaal getal. Hexadecimaal betekent zestientallig. Wat houdt dit in? Er zijn analoge en digitale computers. Bij een analoge machine is een getal de hoogte van een spanning (1=1 Volt, 100=100 Volt, enzovoort). Bij digitale computers is niet de hoogte van de

spanning van belang, maar twee toestanden: wel of geen stroom. Digitaal betekent dat er met cijfers wordt gewerkt. De toestanden AAN en UIT komen overeen met 1 en 0. Bij digitale computers zijn die twee cijfers voldoende om alle waarden weer te geven. Bij vaste opdrachten als de besturing van een machine is een analoge computer soms zinvoller. Maar wanneer het om verschillende problemen gaat, is de digitale computer verreweg superieur. Programmering van een analoge computer is in de ons bekende vorm niet mogelijk. Alle home- en personal computers zijn digitale computers en werken met het tweetallige (of binaire) stelsel (cijfers 0 en 1). Voor de programmeur zijn de volgende talstelsels van belang:

decimale of tientallige stelsel
binaire of tweetallige stelsel
hexadecimale of zestientallige stelsel

Talstelsels zijn ordeningsschema's van cijfers die volgens een bepaald principe functioneren. U kunt elk getal in andere talstelsels omrekenen. In alle stelsels stijgt de waarde van een cijfer van rechts naar links. Nemen we het decimale getal 7345, dan is de volgorde van rechts naar links: eenheden, tientallen, honderdtallen en duizendtallen.

1.3.1 Het decimale stelsel

macht	getal	benaming	afk.
10^0	1	een	E
10^1	10	tien	T
10^2	100	honderd	H
10^3	1.000	duizend	D
10^4	10.000	tienduizend	TD
10^5	100.000	honderduizend	HD
10^6	1.000.000	miljoen	M

Het decimale getal 1335 is

1D + 3H + 3T + 5E of

1*1000 + 3*100 + 3*10 + 5*1 of

1*10^3 + 3*10^2 + 3*10^1 + 5*1^0

Een macht met als exponent 0 is 1. Bijvoorbeeld:

10^0=1
2^0=1
134^0=1

algemeen:

x^0 =1

1.3.2 Het binaire stelsel

Het tweetallige stelsel is opgebouwd volgens hetzelfde principe. De cijfers corresponderen niet met machten van tien maar met machten van twee. De basis van het tweetallige stelsel is 2. Het getal dat aan een talstelsel ten grondslag ligt, noemen we grondtal.

10101101 (binair) = 173 (decimaal)

Die bewering moest maar eens aan een nader onderzoek worden onderworpen. We zetten daartoe het binaire getal en de vermenigvuldigingen met de kwadraten overzichtelijk onder elkaar en tellen vervolgens de produkten op.

```
1       0       1       0       1       1       0       1

1*2^7 + 0*2^6 + 1*2^5 + 0*2^4 + 1*2^3 + 1*2^2 + 0*2^1 + 1*2^0 =

1*128 + 0*64  + 1*32  + 0*16  + 1*8   + 1*4   + 0*2   + 1*1   =

173
```

Het bewijs dat 10101101 inderdaad de tweetallige representatie is van het decimale getal 173, is hiermee geleverd. We kunnen natuurlijk ook decimale getallen omrekenen in binaire. Uitgaande van 173 stellen we ons de vraag welke macht met als grondtal 2 daar nog in kan zitten. In de computerwereld gebruikt men alleen 8-cijferige binaire getallen. De volgende machten met als grondtal 2 kunnen voorkomen:

machten	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
getallen	128	64	32	16	8	4	2	1

In dit geval is dus 2^7=128 de hoogst voorkomende macht met als grondtal 2. Als we 128 van 173 aftrekken, blijft er nog 45 over. We zoeken weer de hoogste macht met als grondtal 2: dat is 2^5=32. 45-32=13 en zo gaan we door tot er geen rest overblijft.

2^3=8 (13-8=5)
2^2=4 (5-4=1)
2^0=1 (1-1=0)

U heeft nu de volgende machten met grondtal 2 berekend: 2^7, 2^5, 2^3, 2^2 en 2^0. De reeks is niet compleet. Als u de machten die voorkomen met 1 kenmerkt en de machten die missen met 0, krijgt u het binaire getal dat overeen komt met 173 in het decimale stelsel:

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	1	0	1	1	0	1

In het vervolg geven we binaire getallen aan door &B aan het getal vooraf te laten gaan.

1.3.3 Bit en byte

Een bit is de kleinste informatie-eenheid, waaruit alle andere informatie is opgebouwd. Bit is de afkorting van BInary digiT (binair cijfer). Bits kunnen 1 of 0 zijn, aan of uit, net als een schakelaar. Een bit die 1 is, is gezet, een bit die 0 is, heet gereset. De MSX heeft een 8-bits processor, dat wil zeggen dat hij binaire getallen met een lengte van 8 bits kan verwerken.

1	0	1	1	0	1	1	1	binair getal
z	r	z	z	r	z	z	z	gezette bits (z); geresette bits (r)
7	6	5	4	3	2	1	0	bitnummers

Elke bit (elk cijfer) van een binair getal heeft een bitnummer. De bit met de laagste waarde (de meest rechtse) heeft het nummer

0. Van rechts naar links lopen de nummers op. De bitnummers corresponderen met de exponenten van de machten met als grondtal 2. Een binair getal van 8 cijfers kan de decimale waarden 0-255 representeren. Dat zijn 256 (2^8) getallen. Acht bits bij elkaar noemt men een BYTE. De computer kan een byte opslaan in een geheugenplaats (adres). Getallen groter dan 255 verdeelt de computer in twee helften, de lowbyte (byte met de laagste waarde) en de highbyte (byte met de hoogste waarde). Deze bytes nemen twee naburige adressen in beslag. U berekent de high- en lowbyte als volgt:

```
getal : 256 = highbyte + rest,
die rest is de lowbyte
```

Een voorbeeld uit de praktijk, het decimale getal 34065.

```
34065 : 256 = 133, rest 17  (34065 = 133 * 256 + 17)

133 = highbyte
 17 = lowbyte
```

De algemene formule in BASIC luidt:

```
highbyte  HB = INT(getal / 256)
lowbyte   LB = getal - HB * 256
```

Op deze manier heeft een getal in het bereik 256 tot en met 65535 (tot 65536 (256*256)) in het geheugen 2 bytes nodig. Om deze binaire getallen te vereenvoudigen is een ander talstelsel nodig.

1.3.4 Het hexadecimale stelsel

Bij het hexadecimale stelsel is het grondtal 16. Bij het decimale stelsel is dat 10 en bij het binaire stelsel 2. Voor de weergave van cijfers met een waarde groter dan 9 gebruikt men in het zestientallige stelsel de letters A-F.

dec.	hex.	dec.	hex.	dec.	hex.	dec.	hex.
0	0	10	A	20	14	30	1E
1	1	11	B	21	15	31	1F
2	2	12	C	22	16	32	20
3	3	13	D	23	17	33	21
4	4	14	E	24	18	..	..
5	5	15	F	25	19	..	..
6	6	16	10	26	1A	..	..
7	7	17	11	27	1B	enzovoort	
8	8	18	12	28	1C		
9	9	19	13	29	1D		

Eerst veranderen we hexadecimale getallen in decimale getallen:

macht	waarde
16^0	1
16^1	16
16^2	256
16^3	4096

```
&H3ABF = 3*16^3 + 10*16^2 + 11*16^1 + 15*16^0
&H3ABF = 3*4096 + 10*256  + 11*16   + 15*1
&H3ABF = 12288  + 2560    + 176     + 15
&H3ABF = 15039
```

Nog een voorbeeld:

```
&H1A3E = 1*16^3 + 10*16^2 + 3*16^1 + 14*16^0
&H1A3E = 1*4096 + 10*256  + 3*16   + 14*1
&H1A3E = 4096   + 2560    + 48     + 14
&H1A3E = 6718
```

Het voordeel van het hexadecimale stelsel is dat u direct de lowbyte en de highbyte kunt aflezen. Voor &H3ABF geldt:

```
highbyte = &H3A

&H3 =  3
&HA = 10

dus:

&H3A =  3*16^1 + 10*16^0 =  58


lowbyte  = &HBF

&HB = 11
&HF = 15
dus:

&HBF = 11*16^1 + 15*16^0 = 191
```

Voer vervolgens deze regel in:

```
PRINT PEEK(&H1D),PEEK(&H1E)
```

Op de adressen &H1D en &H1E staat het adres waarnaar het bedrijfssysteem springt als een routine in het ROM wordt opgeroepen. Een sprongadres kan de waarde 0 tot en met 65535 (&HFFFF) hebben. De computer slaat dit getal op als high- en lowbyte. Het sprongadres berekent u als volgt: met de genoemde BASIC-instructie ziet u op adres &H1D de waarde 23 en op adres &H1E de waarde 2. In het decimale stelsel is het sprongadres

```
535 = 2 * 256 + 23

  2 = &H2
 23 = &H17
dus:

535 = &H217 (highbyte altijd voorop)
```

Een hexadecimaal getal ontleden in high- en lowbyte is dus net zo makkelijk als zo'n getal samenstellen uit die componenten. Gewoonlijk staat de lowbyte van een getal in het laagste geheugenadres, gevolgd door de highbyte. Binaire getallen veranderen in zestientallige is al even simpel. Verdeel een

binair getal in twee blokken van 4 bits. De bits 0-3 heten lownibble en de andere vier highnibble. Elke nibble correspondeert met een hexadecimaal cijfer. Een binair getal van 4 bits kan ten hoogste de waarde 15 hebben (15=8+4+2+1). Alle waarden van 0 tot en met 15 kunt u ook door een hexadecimaal cijfer weergeven (zie boven). Een voorbeeld:

highnibble

1 1 0 1

8 + 4 + 0 + 1 = 13 = &HD

lownibble

1 0 0 1

8 + 0 + 0 + 1 = 9 = &H9

dus: &B11011001 = &HD9 (highnibble voorop)

Met enige oefening weet u welk hexadecimale getal er bij een 4-bits getal hoort en omgekeerd. De volgénde tabel is een aardig geheugensteuntje:

bin.	hex.	dec.
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15

Op dezelfde manier gaat de vertaling van zestientallig naar tweetallig. Elk hexadecimale cijfer wordt vervangen door de

overeenkomstige 4-bits combinatie, bijvoorbeeld &HC7 = &B11000111. Inzicht in de relatie tussen de verschillende talstelsels is de basis voor programmeren in machinetaal.

Opgaven

(1) Maak de volgende tabel af:

dec.	bin.	hex.
130	?	?
?	&B10010011	?
57312	---	?
?	---	&HC0B6
?	?	&H37

(2) Vanaf geheugenplaats &HF000 moet de waarde 37315 worden opgeslagen. Bereken de highbyte en de lowbyte en schrijf de BASIC-instructie die het getal opslaat.

(3) Vanaf geheugenplaats &H0009 staat een belangrijk sprongadres van het bedrijfssysteem. Welke waarde heeft dit adres?

Oplossingen

(1)

dec.	bin.	hex.
130	&B10000010	&H82
147	&B10010011	&H93
57312	---	&HDFE0
49334	---	&HC0B6
55	&B00110111	&H37

(2) highbyte = 145 = &H91; lowbyte = 195 = &HC3
    POKE &HF000,&HC3: POKE &HF001,&H91

(3) lowbyte = PEEK(&H0009), highbyte = PEEK(&H000A)
    sprongadres = &H2683

In de bijlage vindt u een tabel met de waarden van de decimale getallen 0-255 (1 byte) naast die in de twee andere talstelsels die we hebben besproken.

1.4 De geheugenstructuur

Programmeren in machinetaal vereist inzicht in de geheugenstructuur van de computer. MSX-computers hebben een geheugencapaciteit van 64, 48, 32 of 16 Kbyte. 1 Kbyte = 1 kilobyte = 1024 bytes; 64 Kbyte = 64 * 1024 = 65.536 bytes. Een byte bestaat uit 8 bits; in het computergeheugen hebben gegevens de vorm van 8 bits. De 64 Kbyte variant heeft dus 64 * 1024 * 8 bits, dat zijn circa een half miljoen schakelaars die aan- of uitgeschakeld zijn. De 65.536 bytes staan in het RAM van de computer. RAM betekent Random Access Memory (schrijf- en leesgeheugen). De RAM-bytes zijn genummerd van &H0000 tot &HFFFF. We noemen die nummers adressen. Het is normaal om ze als hexadecimale getallen te schrijven. In BASIC komt u met de instructies PEEK en POKE direct in het RAM. PEEK, gevolgd door het adres, leest de waarde van de byte op het aangegeven adres en POKE, gevolgd door een adres en een getal, zet dat getal op die geheugenplaats. Omdat aan elke byte een adres is toegekend en een byte uit 8 bits bestaat, dus in het bereik 0-255 (&H00-&HFF) ligt, mag de waarde die u opslaat alleen in dit bereik liggen. Het adres moet ook tussen &H0000 en &HFFFF liggen. Het RAM dient voor de opslag van BASIC-programma's. Bovendien bevinden zich in het RAM een aantal belangrijke routines van het bedrijfssysteem en informatie over actuele kleuren, toetsenfuncties, tekens die u zelf hebt gedefinieerd, enzovoort. Ondoordachte POKE-instructies hebben tot gevolg dat de computer het laat afweten. We zeggen dan dat hij 'hangt'. POKE &HFF0C,199 is bijvoorbeeld uit den boze. De structuur van het RAM is als volgt:

&HOOOO-&H7FFF	machinetaalprogramma's	(64K versie)
&H8000-&HBFFF	BASIC-programma's	(versies >32K)
&HC000-&HF37F	BASIC-programma's	(alle versies)
&HF380-&HFFFF	bedrijfssyteem	(alle versies)

Met de instructie CLEAR, gevolgd door de grootte van de string en een adres, kunt u de ruimte voor BASIC-programma's begrenzen. Het bereik vanaf het in de CLEAR-instructie genoemde adres tot en met &HF37F kunnen we gebruiken om machinetaalprogramma's op te slaan. Systeemroutines gebruiken iets meer dan 3 Kbyte van het RAM. De componenten waardoor we in BASIC kunnen programmeren, de interpreter en het bedrijfssysteem, bevinden zich niet in het RAM, maar in een ander geheugen, het ROM (Read Only Memory). In de fabriek is het ROM in de computer ingebouwd en volgezet met gegevens en programma's in machinetaal. De MSX heeft twee ROM-bouwstenen van 16 Kbyte (soms 3, als er extra programmatuur voorhanden is, zoals bij Sony), waarvan de adressen die van het RAM overlappen. Dat is niet te vermijden want de Z80 heeft maar 16 adreslijnen. Het adres van een byte kan dus niet langer zijn dan 16 bits. Dat komt precies overeen met &H0000 tot &HFFFF. Als u de computer aanzet, zijn de adressen van &H0000 tot &H7FFF gereserveerd voor het ROM en die van &H8000 tot &HBFFF voor het RAM. In sommige uitvoeringen liggen in dit gebied de extra ROM-routines. Het bovenste blok van 16Kbyte (&HC000 tot &HFFFF) wordt in eerste instantie altijd aan het RAM toegewezen. Als u het onderste RAM wilt lezen, moet u dat eerst aan de CPU meedelen. Daarna kunt u dezelfde adressen aanspreken als in het ROM.

ROM 1	&H0000-&H3FFF	bedrijfssysteem
ROM 2	&H4000-&H7FFF	BASIC
ROM 3	&H8000-&HBFFF	extra programma (indien aanwezig)

Zonder de routines van het bedrijfssysteem kan de computer niet functioneren. Het bedrijfssysteem zorgt ervoor dat de randapparatuur wordt gestuurd, beheert de gegevens, enzovoort. In het onderste ROM-bereik staan ook de kopieën van de systeemroutines uit het RAM. Wordt de computer ingeschakeld of een RESET uitgevoerd, dan kopieert het ROM deze routines in het RAM. In het ROM bevindt zich ook het karaktergeheugen (&H1BBF-&H23BC), waarin elk teken dat de computer kent in een bitmatrix is weergegeven (0 = geen punt, 1 = wel een punt). De programma's in het BASIC-ROM voeren de BASIC-instructies van de MSX uit. De tabel met de instructieset begint bijvoorbeeld op adres &H3A72. De methode om meer dan de beschikbare 64Kbyte te adresseren heet bank switching (bereik omschakelen). De MSX heeft vier banks, in het handboek 'slots' (contactdozen) genoemd. De benaming 'slot' is logisch als u bedenkt dat ook de cartridges via deze procedure

kunnen worden aangesproken. Slot 0 correpondeert met de ingebouwde ROM's (bedrijfssysteem, BASIC en extra programma). In de 16Kbyte uitvoering bevat slot 0 ook nog het 16Kbyte RAM. Dan is het vol, want ieder slot heeft een maximum capaciteit van 64Kbyte. Meestal valt het RAM onder slot 2; slot 1 en 3 corresponderen met de beide cartridges. Elk adresblok van 16Kbyte kan maar aan een van de vier slots worden toegewezen.

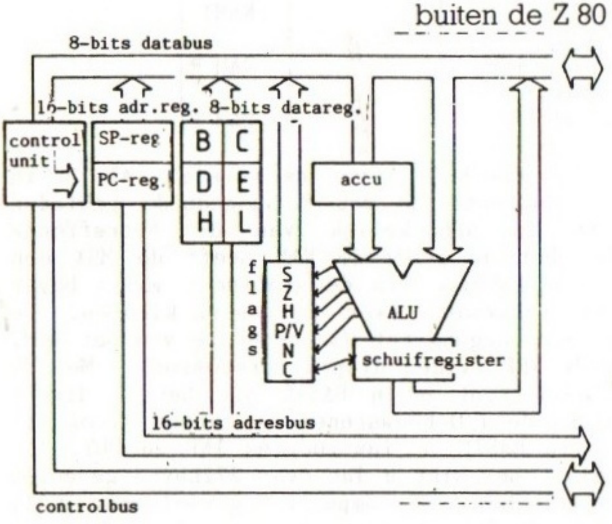
adres	slot 0	slot 1	slot 2	slot 3
&H0000-&H3FFF	bedrijfssysteem * ROM	connector 1	(RAM)	connector 2
&H4000-&H7FFF	BASIC * ROM		(RAM)	
&H8000-&HBFFF	eigen programma ROM		RAM *	
&HC000-&HFFFF	(RAM)		RAM *	

De plaats van het RAM is afhankelijk van de versie. Als u in BASIC programmeert zijn daarvoor de met * aangeduide gebieden gereserveerd. Ook dit is afhankelijk van de betreffende computerversie. Behalve het al genoemde RAM heeft de MSX een apart video-RAM-bereik van 16Kbyte. In de grafische modus bevat dit VRAM de scherminhoud, informatie over sprites en kleuren. De processor heeft niet direct toegang tot dit gedeelte van het RAM. Het wordt beheerd door de VDP (video display processor). Met de instructies VPOKE en VPEEK kunt u in BASIC dit bereik direct aanspreken. Intern spelen de I/O-bouwstenen de eerste viool bij lees- en schrijfacties, in BASIC de instructies INP en OUT. Als alle slots in gebruik zijn, beschikt u dus over 272Kbyte geheugen (inclusief VRAM). Natuurlijk bevat de computer nog veel meer IC's, zoals de Z80 processor of de PSG (programmable sound generator), de geluids-chip. De Z80 processor beschrijven we uitvoerig in het volgende hoofdstuk. Als u meer wilt weten over de diverse chips, raadpleeg dan het boek **MSX Tips en Trucs** van DATA BECKER NEDERLANDS*.

2 DE Z80 PROCESSOR

2.1 De structuur van de CPU

De CPU (central processing unit) van de MSX is de Z80. Dit hoofdstuk behandelt de structuur en de functie van de afzonderlijke onderdelen van deze processor.



afbeelding 1 structuur van de Z80

control unit: controle-eenheid

De CU controleert en stuurt alle processen in de computer.

control bus: controlebus

De controlebus is de lange arm van de CU. Hij stuurt en controleert de perifere bouwstenen, dat zijn de IC's buiten de CPU.

stack pointer: stapelwijzer

Met de SP slaat u gegevens en terugsprongadressen van subroutines op in het RAM. De SP is een 16-bits register, omdat hierin adressen worden opgeslagen. High- en lowbyte van die adressen nemen 2 bytes (16 bits) in beslag.

program counter: programmateller

De PC wijst op het geheugenadres waarop de instructie staat die moet worden verwerkt.

registers B-L

De CPU heeft verschillende registers om gegevens in op te slaan.

flags: vlaggen

Flags markeren bepaalde operaties in de computer. Ze kunnen worden gezet en gewist/gereset.

adresbus (ligt buiten de processor)

De adresbus zorgt voor de verbinding met de andere bouwstenen van de computer. Hij wijst op de geheugenplaats in het ROM of RAM waarvan de inhoud moet worden gelezen of waar iets nieuws moet komen te staan. De adresbus is 16 bits breed, precies genoeg om 64 Kbyte te adresseren.

databus (ligt buiten de processor)

De databus vervoert de gegevens die moeten worden gelezen of opgeslagen (weggeschreven). De adresbus wijst in zo'n geval op het adres van de gegevens. De databus is 8 bits breed.

accumulator

De accumulator (accu) is het belangrijkste register van de CPU.

arithmetical logical unit: rekeneenheid

De ALU voert alle rekenkundige en logische operaties uit en zet flags, afhankelijk van het resultaat.

shift register (schuifregister)

Het schuifregister voert roteer- en schuifoperaties uit.

De registers van de CPU verdelen we in vijf groepen:

accumulator
flags
zes 8-bits registers die gekoppeld kunnen worden
vier 16-bits registers die bij elkaar horen
interrupt-/refresh register

2.2 De accumulator

De accu of het A-register is het belangrijkste register van de Z80. De meeste rekenkundige en logische instructies gebruiken dit register. Elke vergelijkingsinstructie maakt van de accu gebruik. Net als de andere registers (behalve SP, PC, IX en IY) is het A-register een 8-bits register.

2.3 De flags

Het F- of flagregister is 8 bits breed (net als A, B, C, D, E, H en L). De bits in dit register dienen om bepaalde operaties van de ALU te markeren. De bits van het F-register hebben de volgende betekenis:

S	Z		H		P/V	N	C	flag-aanduiding
7	6	5	4	3	2	1	0	bitnummer

C - carry (overdracht)
N - aftrekking
P/V - pariteit/overflow
H - halve overdracht
Z - zero (nul)
S - sign (voorteken)

C-flag (bit 0)

Vindt bij een optelling of aftrekking een overdracht plaats, dan wordt deze bit gezet; zoniet, dan volgt een reset.

N- en H-flag (bit 1, bit 4)

De Z80 gebruikt deze flags intern. Hun betekenis hoeft hier niet te worden uitgelegd.

P/V-flag (bit 2)

Deze flag heeft een dubbele functie: hij wordt gezet in geval van overflow. Verder geeft hij de pariteit (P) van een byte aan.

Z-flag (bit 6)

Deze flag wordt gezet als het resultaat van een aftrekking nul is, anders wordt hij gereset. Is er bij een vergelijking sprake van gelijkheid dan zet de computer deze bit.

S-flag (bit 7)

Is het resultaat van een optelling of aftrekking groter dan 127 dan wordt deze bit gezet. Voor de CPU zijn bytes groter dan 127 negatieve getallen.

Bit 3 en 5 van het flagregister worden niet gebruikt.

2.4 De zes 8-bits registers

Het gaat hier om de registers B, C, D, E, H en L. Deze registers kunnen registerparen vormen als een breedte van 16 bits gewenst is. C, E en L zijn bestemd voor lowbytes en B, D en H voor highbytes.

B/C (byte counter)

Het B-register of registerpaar BC dient vaak als teller voor bijvoorbeeld lussen.

registerpaar DE

Dit is vrij. Het wordt vaak gebruikt om adressen of gegevens voorlopig op te slaan.

H/L (high/low)

In dit registerpaar worden vaak adressen opgeslagen.

Went u alvast aan de benaming van deze registers: sommige instructies gebruiken de registers zoals boven beschreven. Natuurlijk kan ook het L- of het E-register als teller fungeren. Bijzonder aan de Z80 is dat er een tweede set registers is, identiek aan de genoemde. Er kan maar een set tegelijk worden gebruikt.

2.5 De vier 16-bits registers

Het gaat om de registers SP, PC, IX en IY. Bij deze 16-bits registers is het niet mogelijk om high- en lowbytegedeelte apart aan te spreken. Ze kunnen dus niet in twee 8-bits registers worden verdeeld zoals BC, DE en HL. De stack pointer (SP) wijst op een adres met terugsprongadressen of gegevens die daar voorlopig

zijn opgeslagen. Het adres heeft betrekking op een geheugenplaats in het RAM, die deel uitmaakt van de stack (stapelgeheugen). Gegevens op de stack leggen gaat als volgt: als u de computer aanzet, staat de SP op het hoogste adres van de stack (&HF000). Op het moment dat er een byte op de stack moet worden gelegd, verlaagt de computer de SP met 1 en zet de byte op het nieuwe adres. De SP wijst dus steeds de laatste invoer in het stapelgeheugen aan. Als u informatie uit het stapelgeheugen haalt, verloopt het proces omgekeerd. De byte op het adres waarnaar de SP wijst, wordt gelezen en vervolgens wordt de SP met 1 verhoogd. Zo kunt u subroutines op alle mogelijke manieren ordenen. PC is een bijzonder register. Met een programma kunnen er geen wijzigingen in worden aangebracht. De registers IX en IY dienen hoofdzakelijk om (relatieve) adressen op te slaan. Het gebruik van het indexregister komt ongeveer overeen met dat van HL. Het verschil komt in paragraaf 4.1 ter sprake bij de geïndiceerde adressering.

2.6 Interrupt- en refresh-register

Deze registers maken deel uit van de CU (control unit).

register I (interrupt = onderbreken)

Als er een interrupt optreedt, bevat dit 8-bits register de highbyte van het adres waar naartoe gesprongen moet worden. Het onderdeel dat de interrupt heeft opgeroepen, levert de lowbyte.

register R (refresh = opfrissen)

De computer gebruikt dit register als teller om regelmatig de inhoud van het dynamische geheugen op te frissen. Door in korte tijd dezelfde geheugeninhoud steeds opnieuw te laden wordt voorkomen dat informatie verloren gaat.

Als de CPU een instructie uitvoert, gebeurt dat op de volgende manier: de CPU leest de byte op het adres waarnaar de PC wijst en verhoogt daarna de PC met 1 (de PC wijst dan op de volgende byte). De byte wordt als instructie geïnterpreteerd. Vervolgens

leest de processor gegevens die bij de instructie horen en verhoogt de PC weer. De opdracht wordt uitgevoerd en het proces begint van voren af aan.

3 DE INSTRUCTIESET VAN DE Z80

3.1 De invoer van machinetaalprogramma's

Om de instructies van de Z80 direct te kunnen gebruiken, moet u eerst weten hoe een machinetaalprogramma met BASIC wordt ingevoerd en opgeslagen. In BASIC heeft een instructie een regelnummer, in machinetaal een adres.

BASIC		machinetaal		
regelnr.	instructie	adres	instructie	code
9	HL=HL+1	&HF009	INC HL	&H23
10	RETURN	&HF00A	RET	&HC9

Een machinetaalprogramma is een reeks gecodeerde instructies die in opeenvolgende adressen in het geheugen staan. In BASIC zet u met POKE de codes op de betreffende adressen. Het machinetaalprogramma roept u op met USR. Voordien moet u het startadres van het machineprogramma bepalen met DEFUSR. Het startadres is meestal het eerste adres met een machinetaalcode. Reserveer met CLEAR een geheugenbereik om te voorkomen dat er over het machineprogramma heen wordt geschreven. CLEAR 200, &HEFFF reserveert de adressen &HF000-&HF37F. Dat betekent dat u de beschikking heeft over &H380 bytes (1 Kbyte) voor machinetaalprogramma's. Een BASIC-programma dat machinetaalprogramma's laadt, ziet er zo uit:

```
10 CLEAR 200,&HEFFF
20 FOR I=beginadres TO eindadres
30 READ A
40 POKE I,A
50 NEXT I
60 DEFUSR1=beginadres
70 END
80 DATA ...
90 DATA ...
.. .... ...
enzovoort
```

In de DATA-regels staan de codes die het machinetaalprogramma vormen. De variabele "eindadres" moet natuurlijk groter zijn dan

&HEFFF en "beginadres" kleiner dan &HF380. Het geladen programma roept u op met X=USR1. Normaal gesproken gebruiken we &HF000 als startadres.

eindadres = beginadres + programmalengte (in bytes) - 1

De lengte van een programma komt overeen met het aantal getallen in de DATA-regels. In MSX BASIC is het mogelijk met de instructies DEFUSR en USR, gevolgd door een getal, tien verschillende oproepen voor machinetaalprogramma's te definiëren. Voorlopig gebruiken we alleen USR1. De betekenis van het getal achter USR zullen we later toelichten. We zullen dan ook bespreken in hoeverre de USR-instructie een functie is en ook wanneer de oproep variabele=USR1(parameter) of PRINT USR1(parameter) moet luiden. Om kleine programma's in te voeren is de volgende BASIC-routine erg handig:

```
10 CLEAR 200,&HEFFF:CLS
20 INPUT"startadres &H";A$
30 ST=VAL("&H"+A$):IF ST<&HEFFF THEN BEEP:GOTO 20
40 AD=ST
50 IF AD>=&HF380 THEN BEEP: PRINT"Bereik overschreden":END
60 PRINT"Adres &H";HEX$(AD);" : ";
70 INPUT"Waarde &H";A$
80 IF A$="" THEN DEFUSR1=ST:END
90 W=VAL("&H"+A$):A$="":IF W<0 OR W>255 THEN BEEP:GOTO 50
100 POKE AD,W:AD=AD+1:GOTO 50
```

Voer de hexadecimale code direct in. Het programma zorgt ervoor dat er niet hoeft worden gePOKEt. Bij het beginadres hoeft u het kenmerk voor hexadecimale getallen (&H) niet in te voeren. Nu we weten hoe machinetaalprogramma's moeten worden ingevoerd, gaan we ons bezighouden met de instructies van de Z80. Een opmerking: om de machinetaalinstructies te verklaren gebruiken we vaak BASIC-instructies. Wat bijvoorbeeld in machinetaal een register is, is in BASIC een variabele. De naam blijft daarbij gelijk. De instructieset van de Z80 is onderverdeeld in 5 groepen:

overdracht van gegevens
verwerking van gegevens en controles
sprongen
stuurinstructies
in- en uitvoer

3.2 Overdracht van gegevens

Gegevens kunnen worden overgedragen van

register naar register

In BASIC is dat bijvoorbeeld A=B of SP=HL. De machinetaal-instructie heeft het formaat LD A,B (LD = laad).

register naar geheugenplaats

Bij de overdracht van register naar geheugenplaats komt de BASIC-instructie POKE, gevolgd door geheugenadres en variabele (bijvoorbeeld POKE &HF000,HL) overeen met de machinetaalinstructie LD (&HF000),HL.

geheugenplaats naar register

Bij de overdracht van gegevens van geheugen naar register komt bijvoorbeeld LD H,(&HF005) overeen met de BASIC-instructie H=PEEK (&HF005).

3.3 Verwerking van gegevens en tests

De instructies voor de verwerking van gegevens zijn in te delen in vijf groepen:

- rekenkundige operaties (bijvoorbeeld ADD (optellen) en SUB (aftrekken))
- logische operaties (bijvoorbeeld AND, OR)
- telinstructies (INC (increase = verhogen), DEC (decrease = verlagen)
- bitmanipulatie (SET, RES (reset))
- bits verwisselen en verschuiven (R (rotate = draaien), S (shift = verschuiven))

Deze instructies veranderen de register- of geheugeninhoud (RAM).

assembler	BASIC
SUB A,B (Eng.: SUBtraction)	A=A-B
ADD HL,BC (Eng.: ADDition)	HL=HL+BC
AND C	A=A AND C
OR HL	A=A OR PEEK(HL)

Er worden bits in registers of geheugenplaatsen getest (instructie BIT) of de register- of geheugeninhoud wordt vergeleken met de accu (instructie CP (compare)). Afhankelijk van het resultaat zet of wist de ALU de bijbehorende flags in het F-register.

3.4 Sprongen

Met deze instructies kunt u machinetaalprogramma's van sprongen voorzien. We onderscheiden drie soorten sprongen:

- directe sprong naar een 16-bits adres (JP (jump))
- relatieve sprong naar het actuele adres (JR (jump relative))
- subroutinesprongen (CALL (naar subroutine) en RET (terug naar hoofdprogramma))

Een sprong heet voorwaardelijk als de beslissing om te springen afhangt van de status van de flag. Een voorbeeld hiervan is JR NZ,&HF003

assembler	BASIC	
JP	GOTO	
CALL	GOSUB	
RET	RETURN	
JR	----	(vergelijkbaar met een FOR/NEXT-lus)

3.5 Stuurinstructies

Deze instructies kunnen onder andere dienen om een programma te onderbreken of interrupts te sturen.

3.6 In- en uitvoerinstructies

I/O-instructies (input/output) zijn bedoeld om I/O-apparaten te sturen. Afhankelijk van de aangesproken I/O-port kunnen met deze instructies de meest verschillende opdrachten worden uitgevoerd. IC's die vaak worden aangesproken met I/O-instructies zijn:

- PPI programmable peripheral interface
- PSG programmable sound generator
- VDP video display controller (en dus alle randapparatuur: toetsenbord, luidspreker, monitor, printer en cassetterecorder)

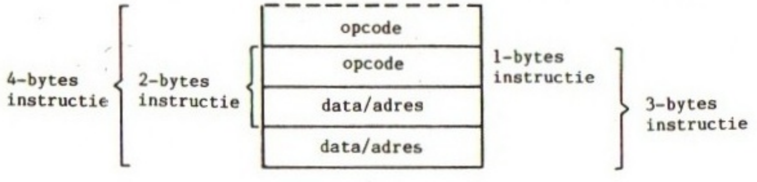
4 DE INSTRUCTIES

4.1 8-bits transferinstructies

In alle 8-bits transferinstructies komt de laadinstructie LD voor. Het algemene formaat is als volgt:

LD doelbereik, oorspronkelijk bereik

De 8-bits transferinstructies laden steeds 8 bits van het oorspronkelijke bereik naar het doelbereik. Aan de hand van deze instructies maken we duidelijk welke manieren de Z80 kent om te adresseren. Elke machinetaalinstructie bestaat uit een operatiecode (opcode), waarop een operand- of adresveld kan volgen. De opcode geeft aan welke operatie wordt uitgevoerd. Soms bevat een opcode bits die als wijzer op een register worden gezet. Hoewel deze bits eigenlijk niet tot de opcode behoren, rekenen we ze er voor het gemak bij. Bij sommige instructies komen na de opcode gegevens- of adresbytes. Bovendien zijn er instructies met een opcode van 2 bytes. Zo kan een instructie een lengte van 1 tot 4 bytes hebben.



afbeelding 2

Om de gegevens of adressen van een instructie te kunnen interpreteren, moet u op de hoogte zijn van de verschillende adresseersystemen.

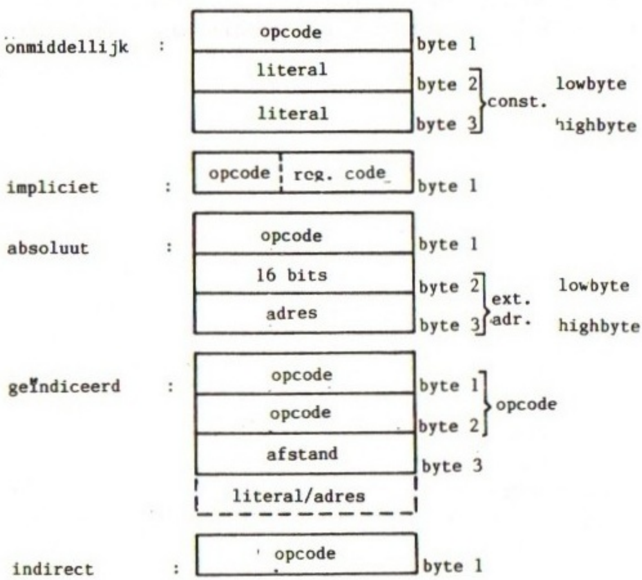
4.1.1 Onmiddellijke adressering (immediate adressing)

Dit is de makkelijkste manier van adresseren. Het formaat is:

```
LD r,n
```

Na de instructie LD komt het register (A, B, C, D, E, H of L) en **onmiddellijk** daarna een 8-bits constante. De constante (ook wel literal genoemd) wordt in het register geladen. Afbeelding 3 geeft dit proces schematisch weer. Op de 8-bits opcode volgt een 8- of 16-bits literal. Bijvoorbeeld:

```
LD C,&H7F        BASIC: C=&H7F
                 (betekent: laad register C met &H7F)
```



afbeelding 3

4.1.2 Impliciete (register)adressering (implied (register) adressing)

Instructies die uitsluitend met registers werken, maken gebruik van impliciete adressering. Formaat:

LD r,r'

Dit betekent: breng de inhoud van het oorspronkelijke register r' naar r (laad r met r'). Registers zijn A, B, C, D, E, H of L. De operands zijn niet gespecificeerd, maar liggen besloten in de registers. De opcode van deze instructie luidt in binaire vorm:

01XXXYYY

Elk van de letters X en Y staat voor een bit. De combinatie XXX betekent doelregister en YYY oorspronkelijk register. De code voor de registers is:

reg.	bin.
A	111
B	000
C	001
D	010
E	011
H	100
L	101

voorbeeld:

LD B,C = 01 000 001 = &H41

Zo geeft u impliciet geadresseerde instructies als 1-bytes opcode weer, wat de snelheid waarmee de instructie wordt afgehandeld ten goede komt. Voorbeeld:

machinetaal	BASIC
LD A,B	A=B

Dit betekent: breng de inhoud van B naar A (laad A met B). Zilog Inc. (de uitvinder van de Z80) bestempelt deze manier van adresseren als registeradressering. Aan die afwijkende definitie van impliciete adressering voldoen alleen de instructies LD I,A;

LD R,A; LD A,R en LD A,I. Wij maken hier geen onderscheid en gebruiken beide termen door elkaar.

4.1.3 Absolute adressering (external adressing)

Absolute adressering noemen we het proces waarbij gegevens uit het geheugen worden gehaald of daar worden opgeslagen. Hierbij wordt het 16-bits adres van de geheugenplaats compleet aangegeven (het absolute adres). Formaat:

```
LD r,(nn) of LD (nn),r
(nn: adres van geheugenplaats)
```

Deze instructie laadt het register (r) met de inhoud van de geheugenplaats (nn) en omgekeerd. Op afbeelding 3 ziet u dat het adres op de opcode volgt. De absolute adressering heeft drie bytes nodig, waardoor deze instructies relatief langzaam zijn. Bijvoorbeeld:

machinetaal	BASIC
LD A,(&HBF93)	A=PEEK(&HBF93)
LD (&HF001),A	POKE &HF001,A

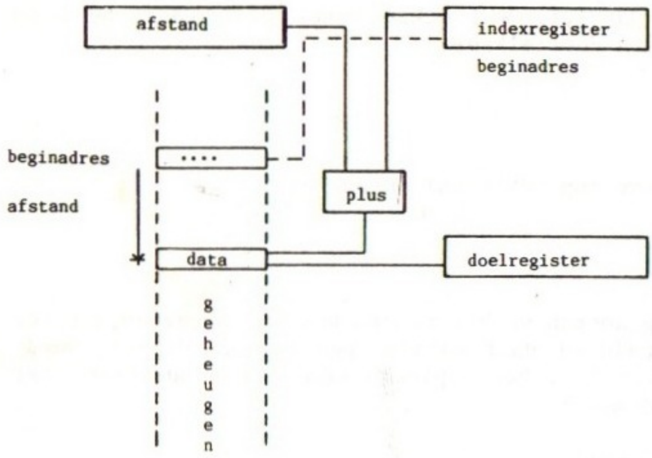
4.1.4 Geïndiceerde adressering (indexed adressing)

Bij de geïndiceerde adressering wordt het adres van de geheugenplaats niet absoluut aangegeven, maar volgt het uit de inhoud van een indexregister en een bepaalde afstand. Formaat:

```
LD r,(XY+d)
LD (XY+d),r

 d : distance (afstand)
 XY: register IX of IY)
```

Het register wordt geladen met de geheugenplaats die het berekende adres heeft en omgekeerd.



afbeelding 4 geïndiceerde adressering: LD r,(XY+d)

Geïndiceerde instructies hebben een 2-bytes opcode, gevolgd door een afstandsaanduiding. De eerste byte van de opcode is:

&HDD bij register IX
&HFD bij register IY

De andere bytes van de code zijn gelijk bij IX en IY. U gebruikt deze manier van adresseren om na elkaar elementen van een blok gegevens aan te spreken. De afstand kan positief of negatief zijn. De afstandsbyte wordt in het twee-complement systeem uitgevoerd (rekenwijze waarbij een negatieve waarde van een getal wordt verkregen door alle bits te inverteren en het resultaat met 1 te vermeerderen). Het indexregister wordt dus steeds verhoogd. Bijvoorbeeld:

machinetaal	BASIC
LD E,(IX+&H32)	E=PEEK (IX+&H32)
LD (IX+&H12),A	POKE IY+&H12,A

4.1.5 Indirecte adressering (indirect adressing)

Deze manier van adresseren lijkt op de voorgaande, alleen adresseert hier een van de registerparen HL, BC of DE de geheugenplaats. Formaat:

```
LD r,(x)
LD (x),r
```

(x: registerparen HL, BC, DE)

Het register wordt geladen met de inhoud van de geheugenplaats die door de inhoud van het registerpaar is geadresseerd. Deze manier van adresseren vereist slechts een 1-bytes instructie. Het register (r) en het registerpaar (x) staan in de opcode. Hierdoor is de instructie sneller en kan toch de complete 64 Kbyte gebruiken. Een voorbeeld:

```
machinetaal   BASIC

LD B,(HL)     B=PEEK (HL)
LD (BC),A     POKE BC,A
```

Hiermee zijn alle manieren om te adresseren besproken die relevant zijn voor 8-bits transferinstructies. Op andere mogelijkheden gaan we later in. In de bijlage vindt u tabellen met alle instructies, gesorteerd naar functie (transfer, sprongen, enzovoort) en adresseringswijze. De opcodes staan daar ook in vermeld. Hieronder staat een tabel met afkortingen voor alle 8-bits laadinstructies. In de bijlage vindt u een tabel met de flagsymbolen. Een toelichting op de twee laatste kolommen (No. of M Cycles en No. of T Cycles) volgt later.

N.B. Alle instructielijsten in dit boek zijn ter beschikking gesteld door Zilog Inc.

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	OP-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T Cycles	Comments	
LD r, r'	r ← r'	•	•	•	•	•	•	01 r r'	1	1	4	r, r'	Reg.
LD r, n	r ← n	•	•	•	•	•	•	00 r 110	2	2	7	000	B
								← n →				001	C
LD r, (HL)	r ← (HL)	•	•	•	•	•	•	01 r 110	1	2	7	010	D
LD r, (IX+d)	r ← (IX+d)	•	•	•	•	•	•	11 011 101	3	5	19	011	E
								01 r 110				100	H
								← d →				101	L
LD r, (IY+d)	r ← (IY+d)	•	•	•	•	•	•	11 111 101	3	5	19	111	A
								01 r 110					
								← d →					
LD (HL), r	(HL) ← r	•	•	•	•	•	•	01 110 r	1	2	7		
LD (IX+d), r	(IX+d) ← r	•	•	•	•	•	•	11 011 101	3	5	19		
								01 110 r					
								← d →					
LD (IY+d), r	(IY+d) ← r	•	•	•	•	•	•	11 111 101	3	5	19		
								01 110 r					
								← d →					
LD (HL), n	(HL) ← n	•	•	•	•	•	•	00 110 110	2	3	10		
								← n →					
LD (IX+d), n	(IX+d) ← n	•	•	•	•	•	•	11 011 101	4	5	19		
								00 110 110					
								← d →					
								← n →					
LD (IY+d), n	(IY+d) ← n	•	•	•	•	•	•	11 111 101	4	5	19		
								00 110 110					
								← d →					
								← n →					
LD A, (BC)	A ← (BC)	•	•	•	•	•	•	00 001 010	1	2	7		
LD A, (DE)	A ← (DE)	•	•	•	•	•	•	00 011 010	1	2	7		
LD A, (nn)	A ← (nn)	•	•	•	•	•	•	00 111 010	3	4	13		
								← n →					
								← n →					
LD (BC), A	(BC) ← A	•	•	•	•	•	•	00 000 010	1	2	7		
LD (DE), A	(DE) ← A	•	•	•	•	•	•	00 010 010	1	2	7		
LD (nn), A	(nn) ← A	•	•	•	•	•	•	00 110 010	3	4	13		
								← n →					
								← n →					
LD A, I	A ← I	•	↕	IFF	↕	0	0	11 101 101	2	2	9		
								01 010 111					
LD A, R	A ← R	•	↕	IFF	↕	0	0	11 101 101	2	2	9		
								01 011 111					
LD I, A	I ← A	•	•	•	•	•	•	11 101 101	2	2	9		
								01 000 111					
LD R, A	R ← A	•	•	•	•	•	•	11 101 101	2	2	9		
								01 001 111					

Notes: r, r' means any of the registers A, B, C, D, E, H, L

IFF the content of the interrupt enable flip-flop (IFF) is copied into the P/V flag

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,

↕ = flag is affected according to the result of the operation.

Opgave

U kent nu de 8-bits laadinstructies. Om ze toe te passen gaan we enkele geheugenplaatsen van het bedrijfssysteem veranderen met behulp van machinetaal. Op adres &HF3BO staat het aantal kolommen van het beeldscherm. In BASIC kunnen we dat aantal instellen met de instructie WIDTH; voor 20 kolommen dus met WIDTH 20. Met een POKE-instructie kunt u de gewenste waarde direct naar het adres sturen. Door POKE &HF3BO,20 wordt het beeldscherm smaller. Het verschil met de WIDTH-instructie is dat bij verandering van de waarde niet automatisch een CLS volgt. Dat kan soms heel handig zijn. Schrijf de BASIC-instructie POKE &HF3BO,20 in machinetaal. Beëindig het programma met de instructie RET (code &HC9).

Oplossing

Eerst een nadere analyse van de vraag. Er moet een getal van 1 byte, dus tussen 0 en 255, worden geladen in adres &HF3BO. We hebben dus een 8-bits laadinstructie nodig. De rest van de oplossing hangt af van de manier van adressering die u kiest. Om een byte in een geheugenplaats te laden, moet natuurlijk eerst het adres van die geheugenplaats worden vastgesteld. De meest rechtstreekse mogelijkheid biedt de absolute adressering, waarbij u het adres compleet aangeeft:

```
LD (&HF3B9),A        (absoluut geadresseerd)
```

De geheugenplaats &HF3B9 krijgt de waarde uit de accu. De gewenste waarde moet dus eerst in de accu zijn geladen.

```
LD A,20              (onmiddellijk geadresseerd)
```

Na elkaar uitgevoerd geven deze instructies het gewenste resultaat.

```
LD A,20
LD (&HF3B9),A
RET
```

RET is het vaste einde van een machinetaalprogramma; de computer schakelt weer over naar BASIC. Tot zover is het programma geschreven in assembler, maar de processor heeft getallen nodig. In de tabellen achter in het boek vindt u voor de onmiddellijk geadresseerde 8-bits laadinstructie LD A,20 de code &H3E,20, waarbij 20 de gewenste waarde is. Bij absolute adressering (LD (&HF3BO),A) krijgen we &H32, &HBO, &HF3. De laatste twee codes staan voor het gewenste adres, eerst de lowbyte (&HBO) en daarna de highbyte (&HF3). De code voor RET is, zoals al eerder gezegd, &HC9. De DATA-regel uit het laadprogramma van paragraaf 3.1 komt er dan als volgt uit te zien:

```
DATA &H3E,20,&H32,&HBO,&HF3,&HC9
```

De lengte van het programma is 6 bytes. De FOR/NEXT-lus loopt dus van &HF000 tot &HF005. RUN de BASIC-lader; het eerste machinetaalprogramma wordt dan in het geheugen gePOKEt. Door de instructie DEFUSR1=&HF000 is het startadres van de instructie USR1 aan het begin van het programma geplaatst. Na invoer van X=USR1(1) start het programma en als alles goed is, krimpt het beeldscherm gelijk ineen tot 20 kolommen. Een andere breedte krijgt u door de tweede waarde in de DATA-regel (hier dus 20) te veranderen en het programma met RUN weer in het geheugen op te slaan.

4.2 16-bits transferinstructies

Ook de 16-bits laadinstructies hebben het formaat

LD doelbereik, oorspronkelijk bereik

De computer stuurt steeds 16 bits over. Deze instructies betreffen de registerparen BC, DE, HL, SP, IX en IY.

4.2.1 Onmiddellijke adressering

Zoals gezegd gaat het om 16-bits registers. Daarom moet de constante na de opcode 16 bits lang zijn. De twee bytes na de opcode bevatten de low- en highbyte van de constante (in deze volgorde). In tegenstelling tot de onmiddellijke adressering met een 1-bytes constante heet deze techniek direct uitgebreide adressering. Formaat:

LD x, nn

x : een 16-bits register (SP, BC, DE, HL, IX, IY)
nn: 16-bits constante

Met deze instructie laadt u register x met de constante nn. Bijvoorbeeld:

machinetaal	BASIC
LD HL,&HF000	HL=&HF000

4.2.2 Impliciete adressering

Er zijn maar drie 16-bits laadinstructies. Ze hebben allemaal betrekking op register SP:

```
LD SP,HL
LD SP,IX
LD SP,IY
```

Deze instructies betekenen: laad het stapelgeheugen met de inhoud van de HL-, IX- of IY-registers. Het equivalent in BASIC is:

```
SP=HL
SP=IX
SP=IY
```

4.2.3 Absolute adressering

De absolute adressering bij de 16-bits instructies bespreken we uitvoeriger. Formaat:

```
LD x,(nn)
LD (nn),x

x: BL, DE, HL, SP, IX of IY
```

Omdat nn op 1 adres wijst (dus maar 1 byte adresseert) en x een 16-bits register is, is het volgende afgesproken: eerst wordt de lowbyte geladen op adres nn in het register en vervolgens de highbyte op nn+1. Een voorbeeld:

```
LD HL,(&HF280)

L-register: lowbyte  uit adres &HF280
H-register: highbyte uit adres &HF281
```

Bij de omgekeerde instructie in de vorm LD (nn),x slaat de computer de lowbyte op in adres nn en de highbyte in nn+1:

```
LD (&HF000),IX

&HF000: lowbyte van IX
&HF001: highbyte van IX
```

Zo'n instructie komt overeen met twee 8-bits laadinstructies. Het volgende voorbeeld maakt dat duidelijk:

```
16 bits              8 bits

LD BC,(&HF005)       LD C,(&HF005) (lowbyte)
                     LD B,(&HF006) (highbyte)
```

Zoals we al eerder hebben opgemerkt luidt de formule om een 16-bits getal in high- en lowbyte op te splitsen:

```
getal = 256 * highbyte + lowbyte
```

Het BASIC-equivalent van de machinetaalinstructie

```
LD DE,(&HF000)
```

is

```
DE = 256 * PEEK(&HF001) + PEEK(&HF000)
```

In het hexadecimale stelsel komt dat neer op

```
DE=VAL("&H"+HEX$(PEEK(&HF001))+HEX$(PEEK(&HF000)))
```

Voor de omgekeerde instructie (bijvoorbeeld LD (&HF100),IY) zijn in BASIC twee instructies nodig:

```
POKE &HF100, IY - INT (IY/256) * 256  (lowbyte)
POKE &HF101, INT (IY/256)             (highbyte)
```

Als deze equivalenten u niet duidelijk zijn, bestudeer dan de paragraaf over de talstelsels nog eens rustig. Substitueer getallen voor DE en IY en voer de berekeningen uit.

Mnemonic	Symbolic Operation	C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
LD dd, nn	dd ← nn	•	•	•	•	•	•	00 dd0 001	3	3	10	dd Pair
								← n →				00 BC
								← n →				01 DE
LD IX, nn	IX ← nn	•	•	•	•	•	•	11 011 101	4	4	14	10 HL
								00 100 001				11 SP
								← n →				
								← n →				
LD IY, nn	IY ← nn	•	•	•	•	•	•	11 111 101	4	4	14	
								00 100 001				
								← n →				
								← n →				
LD HL, (nn)	H ← (nn+1)	•	•	•	•	•	•	00 101 010	3	5	16	
	L ← (nn)							← n →				
								← n →				
LD dd, (nn)	dd_H ← (nn+1)	•	•	•	•	•	•	11 101 101	4	6	20	
	dd_L ← (nn)							01 dd1 011				
								← n →				
								← n →				
LD IX, (nn)	IX_H ← (nn+1)	•	•	•	•	•	•	11 011 101	4	6	20	
	IX_L ← (nn)							00 101 010				
								← n →				
								← n →				
LD IY, (nn)	IY_H ← (nn+1)	•	•	•	•	•	•	11 111 101	4	6	20	
	IY_L ← (nn)							00 101 010				
								← n →				
								← n →				
LD (nn), HL	(nn+1) ← H	•	•	•	•	•	•	00 100 010	3	5	16	
	(nn) ← L							← n →				
								← n →				
LD (nn), dd	(nn+1) ← dd_H	•	•	•	•	•	•	11 101 101	4	6	20	
	(nn) ← dd_L							01 dd0 011				
								← n →				
								← n →				
LD (nn), IX	(nn+1) ← IX_H	•	•	•	•	•	•	11 011 101	4	6	20	
	(nn) ← IX_L							00 100 010				
								← n →				
								← n →				
LD (nn), IY	(nn+1) ← IY_H	•	•	•	•	•	•	11 111 101	4	6	20	
	(nn) ← IY_L							00 100 010				
								← n →				
								← n →				
LD SP, HL	SP ← HL	•	•	•	•	•	•	11 111 001	1	1	6	
LD SP, IX	SP ← IX	•	•	•	•	•	•	11 011 101	2	2	10	
								11 111 001				
LD SP, IY	SP ← IY	•	•	•	•	•	•	11 111 101	2	2	10	
								11 111 001				qq Pair
PUSH qq	(SP-2) ← qq_L	•	•	•	•	•	•	11 qq0 101	1	3	11	00 BC
	(SP-1) ← qq_H											01 DE
PUSH IX	(SP-2) ← IX_L	•	•	•	•	•	•	11 011 101	2	4	15	10 HL
	(SP-1) ← IX_H							11 100 101				11 AF
PUSH IY	(SP-2) ← IY_L	•	•	•	•	•	•	11 111 101	2	4	15	
	(SP-1) ← IY_H							11 100 101				
POP qq	qq_H ← (SP+1)	•	•	•	•	•	•	11 qq0 001	1	3	10	
	qq_L ← (SP)											
POP IX	IX_H ← (SP+1)	•	•	•	•	•	•	11 011 101	2	4	14	
	IX_L ← (SP)							11 100 001				
POP IY	IY_H ← (SP+1)	•	•	•	•	•	•	11 111 101	2	4	14	
	IY_L ← (SP)							11 100 001				

Notes: dd is any of the register pairs BC, DE, HL, SP
qq is any of the register pairs AF, BC, DE, HL
$(PAIR)_H$, $(PAIR)_L$ refer to high order and low order eight bits of the register pair respectively.
E.g. BC_L = C, AF_H = A

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
‡ flag is affected according to the result of the operation.

Opgave

Gebruik 16-bits laadinstructies voor indirecte adressering om het programma uit de vorige paragraaf te herschrijven.

Oplossing

```
LD HL,&HF3B0
LD (HL),20
RET
```

Eerst wordt met een 16-bits laadinstructie voor onmiddellijke uitgebreide adressering het adres in register HL geladen. Vervolgens wordt de waarde in de nu indirect geadresseerde geheugenplaats opgeslagen. De DATA-regel van het omgewerkte programma ziet er als volgt uit:

```
DATA &H21,&HB0,&HF3,&H36,20,&HC9
```

In principe is het ook mogelijk geïndiceerd te adresseren, maar dat is bij deze toepasssing niet erg zinvol vanwege de onnodige verlenging van het programma. Draai hetzelfde programma nog eens, maar nu met adres &HF3DB. Dit adres bepaalt het geluid van het toetsenbord. Bij de waarde 0 is er niets te horen. In alle andere gevallen hoort u bij het aanraken van een toets een klik.

Opgave

Bij 16-bits laadinstructies voor absolute adressering gaat het steeds om twee bytes. We gaan met deze instructies de positie van de cursor veranderen. Die plaats wordt door precies twee bytes bepaald: de X-coördinaat door adres &HF3DD en de Y-coördinaat door

adres &HF3DC. Schrijf een programma waarbij de X-coördinaat 5 en de Y-coördinaat 10 wordt.

Oplossing

Ook hier kunnen we verschillende kanten op. Eerst moet register HL worden geladen met de gewenste waarden. De waarde voor de Y moet in L komen te staan en die voor de X in H omdat de Y op de laagste van de twee plaatsen (&HF3DC) moet staan. We kunnen de registers apart laden:

```
LD L,10
LD H,5
```

of het registerpaar HL in een keer laden. Dan moeten we L (= Y) en H (= X) eerst samenvoegen tot een 16-bits getal.

```
L  =  10          =  &H0A
H  =   5          =  &H05

HL = 256 * 5 + 10 = &H050A
```

Dit getal gaat naar HL.

```
LD HL,&H050A
```

Deze oplossing heeft de voorkeur omdat het de kortste is. De code voor de instructie LD HL,adr is &H21. Rekening houdend met de volgorde lowbyte, highbyte is de complete instructie in machinetaal:

```
&H21,&H0A,&H05
```

Deze waarde moet in de adressen &HF3DC en &HF3DD komen te staan.

```
LD (&HF3DC),HL
```

Net zoals hierboven wordt dat vertaald in

```
&H22,&HDC,&HF3
```

De complete DATA-regel, inclusief RET, luidt

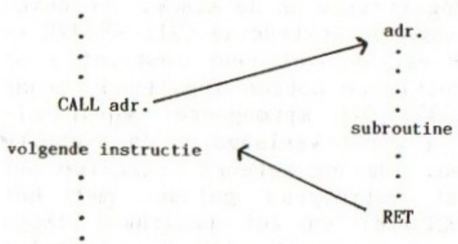
```
DATA &H21,&H0A,&H05,&H22,&HDC,&HF3,&HC9
```

Als u de cursor op het scherm wilt zien (en dat is wel zo handig), voeg dan de volgende regel toe aan het laadprogramma in paragraaf 3.1.

```
11 X=USR1(1):PRINT"*"
```

4.3 Stackinstructies

Om te begrijpen hoe de stack werkt, moet u weten wat er in de Z80 gebeurt als het programma naar een subroutine springt. CALL, gevolgd door een adres, is de assemblerinstructie die daarvoor zorgt. Het voornaamste probleem is dat de CPU het adres van de volgende instructie moet onthouden omdat daar moet worden verdergegaan bij een sprong terug naar het hoofdprogramma (RET).



afbeelding 5 oproep van subroutine

Omdat de registers voor andere belangrijke zaken worden gebruikt, dienen de sprongadressen buiten de CPU (dus in het RAM) te worden opgeslagen. Dat is op die manier helaas maar met 1 sprongadres mogelijk. Subroutines in elkaar schuiven is er niet bij. Daarom moeten we onze toevlucht nemen tot de stack (stapelgeheugen), een bereik in het RAM. Sprongadressen worden opgestapeld en bij een sprong terug neemt het programma het bovenste adres en springt naar de geheugenplaats die de inhoud vormt van dat adres. Dit principe wordt vaak aangeduid met de Engelse term 'first in, last out'. Het proces herhaalt zich totdat de sprongadressen op zijn en u zich weer in het hoofdprogramma bevindt. Het register SP

houdt de hoogte van de stapel bij. De stack pointer wijst steeds op het laatste adres op de stapel. De stapel in de computer staat overigens op z'n kop. Bij het eerste sprongadres hoort het hoogste adres van de stack, het laatste sprongadres correspondeert met het laagste. De stack in de MSX ligt in het bovenste gedeelte van het RAM. De exacte plaats is afhankelijk van de geheugenindeling, zoals vastgesteld met de instructies CLEAR en MAXFILES. Uitgaande van een gedeelte van de stack verloopt de CALL-instructie als volgt:

```
      .....
      .....
      .....
      &HEBF4  registratie
      &HEBF3  registratie
      &HEBF2  registratie
SP-->&HEBF1  laatste registratie
      &HEBF0  ruimte voor nieuwe registratie
      &HEBEF  ruimte voor nieuwe registratie
      .....
      .....
      .....
```

Register SP wijst op de laatste registratie in de stack. We nemen aan dat de processor in het programma de instructie CALL &H077E is tegengekomen op adres &H052D en dat de volgende instructie op adres &H0530 staat. Na de instructie te hebben ingelezen, staat de PC (program counter) op &H530. Dit sprongadres wordt als lowbyte en highbyte opgeslagen. SP wordt verlaagd en de highbyte komt op het nieuwe adres te staan. Daarna gebeurt hetzelfde met de lowbyte. Register PC wordt vervolgens geladen met het startadres van de subroutine (&H077E) en het programma loopt verder. Het relevante gedeelte van de stack ziet er als volgt uit:

```
      ......
      ......
      ......
      &HEBF0    &H05 highbyte van volgende instructie
SP-->&HEBEF    &H30 lowbyte van volgende instructie
      ......
      ......
      ......
```

De SP wijst op de laatste registratie. Bij RET verloopt het proces omgekeerd: de byte op het actuele SP-adres wordt als lowbyte in de PC geladen, de SP wordt met 1 verhoogd en dan is de highbyte van het terugsprongadres aan de beurt. Nadat dat deel

van het adres in register PC is geladen, wordt SP weer verhoogd. Bij dit terugsprongadres aangekomen gaat het programma verder vanaf de plaats die PC aangeeft.

```
     ......
     ......
     ......
SP-->&HEBF1  ...
     &HEBF0  &H07
     &HEBEF  &H83
     ......
     ......
     ......
```

Deze processen verlopen automatisch bij CALL of RET. Zo blijft de volgorde in de stack steeds juist en wijst de SP op de goede plaats. Verandert u register SP direct in het programma, dan kan de volgorde door elkaar raken en hangt de computer. Wees daarom voorzichtig met LD SP,x. Een andere functie van de stack is dat er gegevens in kunnen worden opgeslagen en opgeroepen met de volgende instructies:

```
PUSH (op de stack leggen)
POP  (van de stack halen)
```

PUSH werkt net zo als CALL. Na verlaging van register SP legt de CPU de gegevens op de stack. Bij POP leest hij de gegevens en verhoogt de SP. Met PUSH en POP kan de CPU alle registerparen stapelen behalve SP. Het formaat van PUSH en POP:

```
PUSH qq
POP  qq

qq: AF, BC, DE, HL, IX, of IY
```

Omdat de accu een 8-bits register is en het ook zinvol is register F (flag) op de stack te leggen, worden ze samen behandeld. Iets voorlopig op de stack leggen is nodig als de opslagregisters vol raken. Bijvoorbeeld:

HL bevat het eerste op te tellen getal
BC bevat het tweede op te tellen getal

In het programma wordt een subroutine opgeroepen die HL en BC bij elkaar optelt en het resultaat van de optelling in HL zet. Als u het eerste getal van de optelling later nog nodig heeft, moet u het tijdig op de stack leggen.

```
LD HL, eerste getal
LD BC, tweede getal
PUSH HL
CALL optelling
.....
.....
.....
POP HL
.....
.....
.....
```

Met POP HL haalt u het eerste getal van de stack. Let erop dat de POP-instructie die gerelateerd is aan de PUSH-instructie in dezelfde subroutine moet staan, anders worden de door PUSH opgeslagen gegevens opgevat als terugsprongadres voor RET en hangt de computer. Een vergelijking tussen machinetaal en BASIC leert ons het volgende:

machinetaal	BASIC
PUSH AF	POKE SP-1,A (highbyte) POKE SP-2,F (lowbyte) SP=SP-2
POP BC	BC=PEEK(SP)+256*PEEK(SP+1) SP=SP+2

Omdat PUSH en POP register SP als adreswijzer gebruiken, rekenen we ze tot de indirecte adressering. Een voorbeeld:

(1) PUSH HL

```
SP = &HEB05
HL = &H1234
```

na uitvoering:

```
    &HEB04: &H12  highbyte van HL
SP-->&HBE03: &H34  lowbyte van HL

    SP = &HEB03
    HL = &H1234
```

(2) POP HL

```
SP = &HEB03
HL = &HFFFF
```

na uitvoering:

```
SP = &HEB05
HL = &H1234
```

De lijst met stackinstructies vindt u in paragraaf 4.2 over de 16-bits laadinstructies.

4.4 Wisselinstructies

De Z80 kent behalve de instructies voor enkelvoudige gegevensoverdracht (LD) ook de instructie EX (exchange). Deze verwisselt de inhoud van twee plaatsen. De instructie EX DE,HL verwisselt bijvoorbeeld de inhoud van de registers DE en HL. De EX-instructie met indirecte adressering verwisselt de inhoud van de registers HL, IX of IY en het bovenste stackelement (SP blijft gelijk). Formaat:

```
EX (SP),x

x: HL, IX of IY
```

Verder zijn er wisselinstructies die de inhoud van de eerste en de tweede registerset verwisselen. Bij elk van de registers A, BC, DE, HL en F hoort namenlijk een register A', BC', DE', HL' en F'. U werkt meestal met de eerste reeks registers (A-F). EX AF,AF' verwisselt de inhoud van accu- en flagregister met de registers A' en F'. De instructie EXX verwisselt de registerparen BL, DE, HL en BC', DE', HL'. Deze instructies zijn impliciet geadresseerd. Een voorbeeld:

machinetaal	BASIC
EX DE,HL	SWAP DE,HL
EX (SP),HL	SCH=HL:HL=256*PEEK(SP+1)+PEEK(SP): POKE SP+1,INT(SCH/256):POKE SP, SCH-INT(SCH/256)*256

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
EX DE, HL	DE ↔ HL	•	•	•	•	•	•	11 101 011	1	1	4	
EX AF, AF'	AF ↔ AF'	•	•	•	•	•	•	00 001 000	1	1	4	
EXX	(BC, DE, HL) ↔ (BC', DE', HL')	•	•	•	•	•	•	11 011 001	1	1	4	Register bank and auxiliary register bank exchange
EX (SP), HL	H ↔ (SP+1)	•	•	•	•	•	•	11 100 011	1	5	19	
	L ↔ (SP)											
EX (SP), IX	IX_H ↔ (SP+1)	•	•	•	•	•	•	11 011 101	2	6	23	
	IX_L ↔ (SP)							11 100 011				
EX (SP), IY	IY_H ↔ (SP+1)	•	•	•	•	•	•	11 111 101	2	6	23	
	IY_L ↔ (SP)							11 100 011				
LDI	(DE) ← (HL)	•	•	↕ ①	•	0	0	11 101 101	2	4	16	Load (HL) into (DE), increment the pointers and decrement the byte counter (BC)
	DE ← DE+1							10 100 000				
	HL ← HL+1											
	BC ← BC-1											
LDIR	(DE) ← (HL)	•	•	0	•	0	0	11 101 101	2	5	21	If BC ≠ 0
	DE ← DE+1							10 110 000	2	4	16	If BC = 0
	HL ← HL+1											
	BC ← BC-1											
	Repeat until BC = 0											
LDD	(DE) ← (HL)	•	•	↕ ①	•	0	0	11 101 101	2	4	16	
	DE ← DE-1							10 101 000				
	HL ← HL-1											
	BC ← BC-1											
LDDR	(DE) ← (HL)	•	•	0	•	0	0	11 101 101	2	5	21	If BC ≠ 0
	DE ← DE-1							10 111 000	2	4	16	If BC = 0
	HL ← HL-1											
	BC ← BC-1											
	Repeat until BC = 0											
CPI	A - (HL)	•	↕ ②	↕ ①	↕	1	↕	11 101 101	2	4	16	
	HL ← HL+1							10 100 001				
	BC ← BC-1											
CPIR	A - (HL)	•	↕ ②	↕ ①	↕	1	↕	11 101 101	2	5	21	If BC ≠ 0 and A ≠ (HL)
	HL ← HL+1							10 110 001	2	4	16	If BC = 0 or A = (HL)
	BC ← BC-1											
	Repeat until A = (HL) or BC = 0											
CPD	A - (HL)	•	↕ ②	↕ ①	↕	1	↕	11 101 101	2	4	16	
	HL ← HL-1							10 101 001				
	BC ← BC-1											
CPDR	A - (HL)	•	↕ ②	↕ ①	↕	1	↕	11 101 101	2	5	21	If BC ≠ 0 and A ≠ (HL)
	HL ← HL-1							10 111 001	2	4	16	If BC = 0 or A = (HL)
	BC ← BC-1											
	Repeat until A = (HL) or BC = 0											

Notes: ① P/V flag is 0 if the result of BC-1 = 0, otherwise P/V = 1

② Z flag is 1 if A = (HL), otherwise Z = 0.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,

↕ = flag is affected according to the result of the operation.

4.5 Bloktransfer- en blokzoekinstructies

4.5.1 Bloktransferinstructies

Bloktransferinstructies dragen niet 1 of 2 bytes over, maar hele blokken. De meeste microprocessors kennen ze niet; ze zijn specifiek voor de Z80 en vergroten de prestaties van programma's. De volgende zaken kenmerken een blok gegevens:

- begin- of eindadres van het blok wordt in HL opgeslagen
- en de lengte van het blok in bytes in BC (byte counter)

Hiermee definieert u blokken tot een lengte van 64 Kbyte, die op een willekeurige plaats (HL) in het geheugen beginnen. Voor de overdracht moet u nog begin- of eindadres van het doelblok aangeven. Dit wordt in DE opgeslagen. Als u deze gegevens in de registers heeft gezet, begint de eigenlijke bloktransferinstructie. Er zijn er vier.

```
LDD
LDDR
LDI
LDIR
```

Elke bloktransferinstructie verlaagt de teller BC na overdracht van een byte. LDI en LDIR verhogen de wijzers HL en DE, die dan op bron- en doeladres wijzen van de volgende byte die moet worden overgedragen. LDD en LDDR verlagen de wijzer. Voor deze instructies laadt u eerst HL en DE met het oorspronkelijke adres of het doeladres van het blok. De R aan het eind van de instructie staat voor repeat. Het programma herhaalt deze instructies net zolang tot BC=0, dus tot het hele blok is overgedragen. Nu de instructies apart.

LDI laad en verhoog (incrementeer)

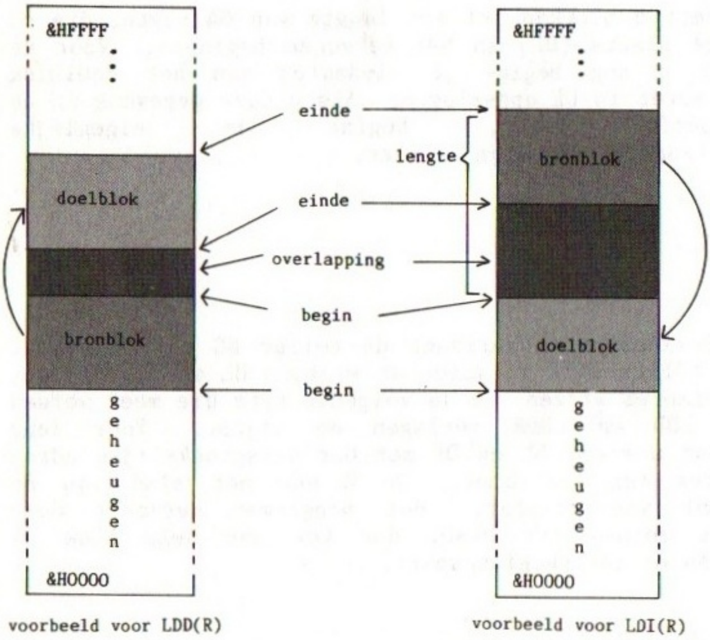
Deze instructie brengt een byte van adres HL naar adres DE en verlaagt daarna BC. De adreswijzers HL en DE worden verhoogd en alles is klaar voor een eventuele voortzetting van de overdracht. Daarvoor dient deze instructie:

LDIR laad, verhoog (incrementeer) en herhaal (repeat)

De overdracht verloopt als bij LDI. Bovendien komt de PC automatisch terug bij deze instructie en herhaalt het proces tot BC=0. Daarna brengt de volgende instructie het programma weer op gang.

LDD laad en verlaag (decrementeer)

LDD functioneert op dezelfde manier als LDI, alleen begint de overdracht van het blok bij het eindadres; HL en DE worden dus verlaagd. Dit verschil is belangrijk als doelblok en oorspronkelijk blok elkaar overlappen. Als u hier de verkeerde instructie gebruikt, schrijft u over de gegevens van het oorspronkelijke blok heen, al voordat ze zijn overgedragen.



afbeelding 6 bloktransferinstructies

LDDR laad, verlaag en herhaal

Als LDD, maar herhaalt de instructie tot het hele blok is overgebracht. Een voorbeeld:

machinetaal	BASIC
LDIR	10 POKE DE,PEEK(HL) 20 HL=HL+1 30 DE=DE+1 40 BC=BC-1 50 IF BC<>0 THEN 10
LDD	POKE DE,PEEK(HL): DE=DE-1:HL=HL-1:BC=BC-1

Aan de grootte van het BASIC-programma ziet u dat de machinetaalinstructie erg handig is.

Flagbeïnvloeding: is BC na de uitvoering =0, dan is P/V=0. De repeat-instructies LDDR en LDIR zetten de P/V steeds op 0.

4.5.2 Blokzoekinstructies

Blokzoekinstructies doorzoeken een blok gegevens op een bepaalde byte. Eerst slaat de instructie de byte waar het om gaat op in de accu. Is de byte gevonden, dan wordt de Z-flag gezet en worden de repeatinstructies gestopt. De registers hebben dezelfde functies als bij de bloktransferinstructies.

- HL begin- of eindadres van blok
- BC byte counter: lengte van blok
- DE heeft geen functie
- A accu: bevat de byte die wordt gezocht

CPIR vergelijkt elke keer de inhoud van HL met die van de accu. Vervolgens verhoogt deze instructie HL en verlaagt BC. Is BC=0, dan wordt de P/V-flag gewist, anders gezet. Zijn A en (HL) gelijk dan verschijnt de Z-flag. Zo niet dan wordt die gereset. De S-flag is net als bij CP de 7e bit van A-(HL). Dit beïnvloedt de carry niet. We onderscheiden vier blokzoekinstructies:

CPI
CPIR
CPD
CPR

De werking komt overeen met die van de bloktransferinstructies. Alle blokinstructies bestaan uit 2 bytes en hun eerste opcodebyte is &ED. Programmeren met blokinstructies betekent meestal ruimte- en tijdwinst. De lijst met bloktransfer- en blokzoekinstructies staat op het eind van paragraaf 4.4.

Opgave

Om de LDDR-instructie in de vingers te krijgen gaan we er wat mee experimenteren. De opdracht is de oorspronkelijke functie te herstellen van toetsen die door gebruik van de KEY-instructie zijn veranderd. Om toetsfuncties te kunnen veranderen moeten de actuele toetsfuncties zijn opgeslagen in het RAM. De oorspronkelijke functies moeten in het ROM aanwezig zijn, zodat die bij inschakeling van de computer direct voorhanden zijn. Het blok met de ROM-kopie begint op adres &H13A9, en het blok met de actuele toetsfuncties op &HF87F. Omdat iedere functietoets een definitie van maximaal 16 tekens kan hebben, is de bloklengte 10*16=&HA0 bytes. Schrijf een machinetaalprogramma die de oorspronkelijke functies van het ROM naar het RAM overzet.

Oplossing

bronblok	vanaf &H13A9	(=register HL)
doelblok	vanaf &HF87F	(=register DE)
bloklengte	&HA0	(=register BC)

We gebruiken de instructie LDIR omdat we de beginadressen van de te verplaatsen blokken kennen. Dit resulteert in het volgende assemblerprogramma:

```
LD HL,&H13A9
LD DE,&HF87F
LD BC,&H00A0
LDIR
RET
```

Omgezet in machinetaal komen de volgende hexadecimale getallen in de DATA-regels van de BASIC-lader te staan:

```
DATA &H21,&HA9,&H13,&H11,&H7F,&HF8
DATA &H01,&H00,&HA0,&HED,&HB0,&HC9
```

Het startadres is &HF000 en het eindadres &HF00B. U kunt nu de oorspronkelijke functies veranderen met KEY. Laad het programma met RUN en start het met X=USR1(1). Om het resultaat waar te kunnen nemen moet u een CLS uitvoeren. Bloklaadinstructies kunt u ook gebruiken om een heel blok gegevens te wissen. We moeten ze dan met opzet verkeerd gebruiken: eerst slaan we een nulbyte op in &HF87F. Vervolgens verschuiven we het blok van &HF87F tot &HF91F (HF87F+&HA0) naar &HF880. Eigenlijk moet u LDDR gebruiken, omdat de bereiken elkaar bij het eindadres van het oorspronkelijke blok overlappen. Gebruikt u LDIR, HL=&HF87F, DE=&HF880, BC=&HA0 dan schrijft het programma steeds de waarde van de geheugenplaats die het laatst is doorgegeven in de geheugenplaats die moet worden overgebracht. Omdat &HF87F de waarde 0 heeft, worden alle bytes van dit blok ook 0. Het complete programma ziet er zo uit:

adres	code	BASIC-regel	assembler-instructie
F000	217FF8	10	LD HL,&HF87F
F003	3600	20	LD (HL),0
F006	1180F8	30	LD DE,&HF880
F008	01A000	40	LD BC,&HA0
F00B	EDB0	50	LDIR
F00D	C9	60	RET

Verklaring:

Elk adres wordt verhoogd met het aantal bytes waaruit de code van het vorige adres bestaat. De code bij &HF000 bestaat uit 3 bytes: &H21, &H7F en &HF8. Daaruit volgt dat het volgende adres &HF003 is. Het aantal codes is maatgevend voor de lengte van de instructie. De assemblerinstructies staan achter de code. Daar gaan we later op in. Oefen nog wat met de bloktransferinstructies. Gebruik verschillende waarden voor HL, DE en BC. Let erop dat het doelblok niet buiten het bereik &H0000-&HFFFF valt, want dan wordt er over systeemroutines heen geschreven en hangt de computer.

4.6 Rekeninstructies

De eerste digitale computers dienden voornamelijk als rekenmachine. Hoewel er veel is veranderd, werken de rekeninstructies vandaag de dag in wezen nog hetzelfde. We wijden ons eerst aan optellen (in machinetaal ADD) en aftrekken (SUB). We laten eerst zien hoe de computer deze operaties uitvoert in het binaire stelsel.

4.6.1 Optellen

In het decimale stelsel tellen we twee onder elkaar staande cijfers bij elkaar op. We noteren de eenheden en brengen de eventuele tientallen over naar de optelling van de cijfers in de volgende kolom. Bijvoorbeeld:

```
  3573
  7154 +
 -----
 10727
```

Hoe dit gaat weten we allemaal nog van de lagere school: 3+4=7, 7+5=12 (2 opschrijven, 1 onthouden), 1+1+5=7 en 7+3=10 (deze 10 wordt als vanzelfsprekend genoteerd, maar de procedure is precies dezelfde als bij 7+5: de 0 wordt genoteerd en de 1 onthouden; maar omdat daarna de optelling afgelopen is, laten we deze redenering in de praktijk achterwege). Er vindt een overdracht plaats zodra de som van twee cijfers groter is dan 9 (10-1); de tientallen spelen een rol in de volgende kolom. In het tweetallige stelsel vindt een overdracht plaats als de som van twee getallen groter is dan 1 (2-1).

Regels:

```
0 + 1 = 1
1 + 0 = 1
0 + 0 = 0
1 + 1 = 0  (bij deze berekening moet u 1 onthouden)
```

Toepassing:

```
1 0 0 1 0 1 1 0 = &H96 = 150
0 0 1 1 1 0 0 1 = &H39 =  57 +
---------------   ----   ---
1 1 0 0 1 1 1 1 = &HCF = 207
    * *
```

* betekent 1 onthouden: 1 overgedragen naar volgende kolom

Het hexadecimale stelsel werkt in principe hetzelfde: een overdracht vindt plaats als het resultaat groter is dan 15.

```
0 0 1 0 1 1 1 0 = &H2E = 46
0 0 0 1 0 1 1 1 = &H17 = 23 +
---------------   ----   --
0 1 0 0 0 1 0 1 = &H45 = 69
```

&HE+&H7 = 14+7 = 21 = &H15 (5 noteren, 1 onthouden)

In dit voorbeeld hebben we te maken met de volgende eigenaardigheid:

```
 11
 11 +
---
110
```

Bij de optelling in de tweede kolom geldt de volgende regel:

1 + 1 + 1 = 1 **en** 1 onthouden

Opgaven

```
(1)   1 0 1 0 1 1 1 0 = &H.. = ...
      0 0 1 0 1 1 1 1 = &H.. = ...
      ---------------   ----   ---
      . . . . . . . . = &H.. = ...
```

```
(2)   0 0 1 1 1 1 1 1 = &H.. = ...
      0 0 1 0 1 1 1 1 = &H.. = ...
      ---------------   ----   ---
      . . . . . . . . = &H.. = ...
```

```
(3)   1 1 1 1 1 1 1 1 = &H.. = ...
      1 1 0 0 1 0 1 0 = &H.. = ...
      ---------------   ----   ---
      . . . . . . . . = &H.. = ...
```

Oplossingen

```
(1)   1 0 1 0 1 1 1 0 = &HAE = 174
      0 0 1 0 1 1 1 1 = &H2F =  47
      ---------------   ----   ---
      1 1 0 1 1 1 0 1 = &HDD = 221

(2)   0 0 1 1 1 1 1 1 = &H3F =  63
      1 0 0 1 1 1 0 1 = &H9D = 157
      ---------------   ----   ---
      1 1 0 1 1 1 0 0 = &HDC = 220

(3)   1 1 1 1 1 1 1 1 = &HFF  = 255
      1 1 0 0 1 0 1 0 = &HCA  = 202
      ---------------   ----    ---
    1 1 1 0 0 1 0 1 1 =&H1C9  = 457
```

Ad (3)

Bij deze optelling vindt een overdracht plaats van kolom 8 (bit 7) naar kolom 9 (bit 8). Een byte heeft echter maar 8 bits. Daarom wordt deze overdrachtsbit, de carry, in bit 0 van het flagregister opgeslagen.

4.6.2 Aftrekken

Aftrekken gaat in het binaire stelsel hetzelfde als in het decimale stelsel. De volgende regels gelden:

```
0 - 1 = 1   1 onthouden
1 - 0 = 1
0 - 0 = 0
1 - 1 = 0
```

Een voorbeeld:

```
0 1 1 0 1 1 1 0 = &H6E = 110
0 0 1 1 0 1 0 1 = &H35 =  53 -
---------------   ----   ---
0 0 1 1 1 0 0 1 = &H39 =  57
    * *       *
```

We kunnen de speciale regels afleiden om met de overdracht verder te rekenen:

```
1 - (1 + 1) = 1   1 onthouden
0 - (1 + 1) = 0   1 onthouden
```

Opgaven

Voer de optellingen van de vorige opdrachten als aftrekkingen uit. Controleer de resultaten zelf door ze om te zetten in het decimale stelsel.

Ad (2)

De uitkomst is een negatief getal: 63-157=-84. Binair:

```
  0 0 1 1 1 1 1 1
  1 0 0 1 1 1 0 1 -
-----------------
1 1 0 1 0 0 0 1 0 = &H1A2
```

Dit is duidelijk een verkeerd resultaat. Negatieve getallen leveren bij aftrekkingen in het binaire stelsel problemen op. Daar is het volgende op gevonden: de computer gebruikt de 7e bit van een binair getal om het voorteken kenbaar te maken (0 is een positief en 1 een negatief getal). Het gevolg is dat het bereik van een byte zich uitstrekt van -128 tot +127. Binaire getallen van elkaar aftrekken is hetzelfde als getallen met voortekens

optellen, bijvoorbeeld: 5-2=5+(-2). De weergave met voortekens, die bij het aftrekken wordt gebruikt, heet twee-complement. Bij twee-complementaire weergave blijven positieve getallen onveranderd, bijvoorbeeld:

```
  5 = &B00000101
126 = &B01111110
```

&B: binair getal

Bij een negatief getal wordt eerst het complement berekend, dat is het binaire getal waarbij alle bits precies tegengesteld zijn gezet: 0 wordt 1 en 1 wordt 0. Het zo verkregen getal heet een-complement of gewoon complement. Een voorbeeld:

```
getal     : 7 = &B00000111
complement:     &B11111000
```

Om het twee-complement van het getal te krijgen, moet het complement met 1 worden verhoogd:

```
&B11111000
         1 +
----------
&B11111001
```

Dit binaire getal is -7 in het twee-complementsysteem. Voor de duidelijkheid recapituleren we het voorafgaande even:

- een positief getal blijft onveranderd
- van een negatief getal wordt het complement berekend en daar wordt 1 bij opgeteld

Een aantal voorbeelden:

dec.	2-comp.
+127	&B01111111
+126	&B01111110
+125	&B01111101
...	....
...	....
...	....
+ 2	&B00000010
+ 1	&B00000001
0	&B00000000
- 1	&B11111111
- 2	&B11111110

```
- 3     &B11111101
...     ........
...     ........
...     ........
-126    &B10000010
-127    &B10000001
-128    &B10000000
```

Om de waarde van een twee-complementaire weergave te achterhalen, vormen we hier weer het twee-complement van. Bijvoorbeeld:

```
&B00001000 = 8

&B11110111   complement
         1 +
----------
&B11111000 = -8  twee-complement

&B00000111   complement
         1 +
----------
&B00001000 = 8  twee-complement
```

De waarde van &B11111000 is dus -8. Als u zo'n getal ziet en het is bekend dat het om een twee-complement gaat dan hoeft u alleen maar het twee-complement van dat twee-complement te berekenen en een minteken voor de uitkomst te zetten. Twee keer het twee-complement levert het oorspronkelijke getal op. De Z80 geeft instructies om de accu-inhoud in het complement (CPL) en het twee-complement (NEG) te veranderen. De functie van deze instructies bekijken we aan de hand van de BASIC-equivalenten. Eerst bekijken we de complementvorming. A bevat een getal tussen 0 en 255 (1 byte). De BIN$-instructie verandert het getal in een string die overeenkomt met het binaire getal. We berekenen bit voor bit het complement van deze string.

```
10 A=&B11011
20 AB$=RIGHT$("0000000"+BIN$(A),8)
30 PRINT"binair getal: ";AB$
40 FOR I=0 TO 7
50 BI$=MID$(AB$,8-I,1):REM Bit Nr I
60 IF BI$="1" THEN BI$="0" ELSE BI$="1"
70 ACPL$=BI$+ACPL$:REM ACPL$ is het complement
80 NEXT
90 PRINT"complement:   ";ACPL$
100 A=VAL("&B"+ACPL$)
```

Regel 50 haalt steeds de i-de bit uit AB$. Regel 60 vormt het

carry gevormd. Dat negeren we eenvoudig. Rekenen met voortekens betekent dat we rekening moeten houden met een aantal bijzondere gevallen. De flags spelen daarbij een belangrijke rol.

Opgave

Bereken het twee-complement van:

- 60
-120
+ 5
- 6

Oplossing

&B11000100 (=196)
&B10001000 (=136)
&B00000101 (= 5)
&B11111010 (=250)

4.6.3 8-bits reken- en telinstructies

Voor optellen en aftrekken zijn er in assembler twee instructies:

ADD;ADC
SUB;SBC

De instructies die op C (=carry) eindigen, houden steeds rekening met de carry-flag. Bit 0 van register F (de carry) wordt opgeteld (ADC) of afgetrokken (SBC). De operanden van deze instructies

complement van de bit, dus 0 wordt 1 en 1 wordt 0. Regel 70 verzamelt de gecomplementeerde bits in ACPL$. Dit programma is erg langzaam. Complementeren gaat in BASIC sneller met XOR. Hoe deze logische instructie functioneert, wordt in de volgende paragraaf verklaard.

```
10 A=&B11011
20 AB$=RIGHT$("0000000"+BIN$(A),8)
30 PRINT"binair getal: ";AB$
40 A=A XOR 255
50 ACPL$=RIGHT$("0000000"+BIN$(A),8)
60 PRINT"complement: ";ACPL$
```

Regel 40 voert de eigenlijke complementvorming uit.

NEG verandert een positief in een negatief getal door het twee-complement te berekenen. Dat ziet er in BASIC zo uit:

```
45 A=A+1
```

Deze regel moet u invoegen in het voorgaande programma.
Bekijk daarna het effect van de volgende regel

```
100 GOTO 40
```

Na het programma te hebben onderbroken zult u zien dat tweemaal de vorming van het twee-complement weer leidt tot het oorspronkelijke getal. Bij twee-complementaire weergave kunt u twee getallen van elkaar aftrekken beschouwen als een optelling van het ene getal met het twee-complement van het andere. Verder is, als bit 7 is gezet (voorteken), het verschil een negatief getal. Een voorbeeld:

```
120-63=57
120=&B01111000
 63=&B00111111
```

Het twee-complement van 63 is &B11000001. Tel op:

```
 01111000    120
 11000001 +  twee-complement van 63
---------
100111001
```

We laten de overdracht van bit 7 naar bit 8 (carry) eerst even buiten beschouwing. Het resultaat is correct: &B00111001 = 57 (120-63). Bit 7 is niet gezet, dus is het resultaat positief. Daarom zou de carry eigenlijk niet gezet mogen worden. Maar omdat we met twee-complement rekenen, wordt ook het complement van de

hebben het formaat:

```
A,x
```

x staat voor r, n, (HL), of (XY+d). We hebben te maken met de volgende soorten instructies:

```
A,r         - impliciet
A,n         - onmiddellijk
A,(HL)      - indirect
A,(XY+d)    - geïndiceerd
```

SUB-instructies hebben als operand alleen r, n, (HL) of (XY+d). A wordt weggelaten omdat dit soort instructies altijd betrekking heeft op de accu. Het zijn 8-bits operaties. De Z80 kent ook 16-bits rekeninstructies. Daarover meer in de volgende paragraaf.

carry-flag

De carry wordt gezet als er een overdracht plaatsvindt van bit 7 naar bit 8. Omdat een byte uit de bits 0-7 bestaat, is de overdracht opgeslagen in de C-flag. Is dit niet het geval, dan wordt de carry-flag gereset.

N- en H-flag

Deze flags worden wel beïnvloed, maar hebben hier geen betekenis.

P/V- of overflow-flag

Er is sprake van overflow als

- er een interne overdracht plaatsvindt van bit 6 naar bit 7, maar geen overdracht van bit 7 naar bit 8 (de carry-flag markeert dat soort (externe) overdracht)
- er in plaats van een interne een externe overdracht plaatsvindt

Het programma zet de flag als bij een rekenkundige handeling het voorteken van de uitkomst (bit 7) ten onrechte wordt veranderd. De V-flag wordt gezet als er een overflow plaatsvindt. Is dit niet het geval, dan wordt deze gereset.

zero-flag

Deze flag wordt alleen gezet als de uitkomst van een berekening 0 is.

sign-flag

Deze flag correspondeert met bit 7 van de uitkomst. Bij de weergave van getallen met voorteken is dit het voorteken; vandaar sign-flag. In de bijlagen staat een tabel met de invloeden van de verschillende instructies op de flags. Voor de verklaring van instructies gebruiken we in het vervolg onderstaande coderingen om de status van een flag na de operatie aan te geven:

1	flag gezet na operatie
0	flag gereset na operatie
|	flag zetten of wissen afhankelijk van resultaat
P	P/V flag: pariteit
V	P/V flag: overflow
	(spatie) geen invloed
X	flag onbekend na operatie

Een voorbeeld:

flags	code	
S	X	onbekend
Z	|	resultaat 0, dan Z=1 en omgekeerd
P/V	1	overflow
C		geen invloed

BASIC-equivalenten van de assemblerinstructies:

assembler	BASIC
ADD A,H	A=A+H
ADC A,&HA9	A=A+&HA9+CF

CF is de carry-flag, de waarde ervan wordt opgeteld:

SUB A,(HL)	A=A-PEEK(HL)
SBC A,L	A=A-L-CF

Nog een paar voorbeelden:

ADD A,(HL)

```
     A =&H1F
     HL=&HB1C9
```

geheugenplaats &HB1C9: &H43

```
     &H1F =  0 0 0 1 1 1 1 1
     &H43 =  0 1 0 0 0 0 1 1 +
             ---------------
             0 1 1 0 0 0 1 0
           8 7 6 5 4 3 2 1 0  bitnummers

bit 8 = 0                   -->  carry-flag = 0
bit 7 = 0                   -->  sign-flag  = 0
resultaat <> 0              -->  zero-flag  = 0
geen ext./int. overdracht   -->  overflow-flag (P/V) = 0
```

accu-inhoud na operatie: &B011000110= &H62

ADD A,D

```
     A=&HE1
     D=&HA2

     &HE1  =   1 1 1 0 0 0 0 1
     &HA2  =   1 0 1 0 0 0 1 0 +
     ----    -----------------
    &H183  = 1 1 0 0 0 0 0 1 1
             8 7 6 5 4 3 2 1 0  bitnummers


bit 8 = 1                -->  carry-flag = 1
bit 7 = 1                -->  sign-flag  = 1
uitkomst <> 0            -->  zero-flag  = 0
ext./int. overdracht     -->  overflow-flag (P/V) = 0
```

accu-inhoud na operatie: &B10000011=&H83

U ziet dat de inhoud van de accu niet het juiste resultaat bevat. Pas als u de carry-flag als 8e bit erbij neemt, krijgt u het juiste resultaat. Bekijk daarom na rekenkundige operaties welke status de flags hebben, zodat u eventuele fouten kunt corrigeren. Bij elke optelling die 256 als uitkomst heeft, zet de computer de zero-flag, hoewel het resultaat niet nul is.

ADC A,&H19

```
    A=&H5A

carry-flag = 1 (gezet)

    &H5A = 0 1 0 1 1 0 1 0
    &H19 = 0 0 0 1 1 0 0 1 +
    ----   ---------------
    &H74 = 0 1 1 1 0 1 0 0

flags:

    S     0
    Z     0
    V     0
    C     0
```

accu-inhoud na operatie: &B01110100=&H74

N.B.: is voor ADC de carry gewist, dan is de instructie gelijk aan ADD.

SUB A,(HL)

```
    A      =&H36
    HL     =&HBC19
    &HBC19=&H15

      0 0 1 1 0 1 1 0    &H36
      1 1 1 0 1 0 1 1 +  twee-complement van &H15
    -----------------
    1 0 0 1 0 0 0 0 1

bit 7 = 0        -->  sign-flag  = 0
bit 8 = 1        -->  carry-flag = 0
```

Bedenk wel dat hier het complement van de werkelijke carry is genomen (uitzondering).

```
geen overflow  -->  V = 0
uitkomst <> 0  -->  Z = 0
```

accu-inhoud na operatie: &B00100001=&H21

SBC A,B

```
A =&H57
B =&H73
CF=1

  0 1 0 1 0 1 1 1    &H57
  1 0 0 0 1 1 0 1    twee-complement van &H73
  1 1 1 1 1 1 1 1 +  twee-complement van &H1(CF)
-----------------
1 1 1 1 0 0 0 1 1
```

flags:

```
S    1
Z    0
V    0
C    1
```

accu-inhoud na operatie: &B11100100=&HE4

De accu-inhoud is het twee-complement van 29. De uitkomst is dus -29 (87-115-1=-29).

Behalve berekeningen uitvoeren in het binaire stelsel is er nog een andere manier om getallen in de computer te verwerken. Hierbij correspondeert elk decimaal cijfer met een blok van 4 bits (BCD, binary coded decimal). Dit is een handige manier als het aantal posities vaststaat en precisie wordt verlangd. In MSX-BASIC worden getallen opgeslagen met enkele, of, in BCD, met dubbele precisie. Voor BCD-operaties is er de speciale instructie DAA, die de accu-inhoud voorbereidt op deze operaties (zie paragraaf 5.4). Bovendien zijn er nog de speciale instructies CPL en NEG. CPL levert het complement van de accu-inhoud en NEG negeert, dat wil zeggen verandert deze in een twee-complement. Sommige normale instructies zijn ook aangepast: SUB A wist bijvoorbeeld de accu. Deze instructie is bijna twee keer sneller dan LD A,0 en de helft korter. Tot deze instructies behoren ook de telinstructies. Ze verlagen of verhogen de waarde van een geheugen. Bij telinstructies kunt u gebruik maken van impliciete, register- en geïndiceerde adressering. Om lussen te programmeren komen ze goed van pas. De werkwijze is eenvoudig:

```
INC x     verhoogt x
DEC x     verlaagt x
```

x kan betekenen: r, (HL), (XY+d)

```
assembler     BASIC

INC r         r=r+1
DEC (HL)      POKE HL,PEEK(HL)-1
```

Sign-, zero- en V-flag worden afhankelijk van het resultaat gezet of gereset. De carry verandert niet. Belangrijk is dat alleén 8-bits telinstructies de flags beïnvloeden.

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
ADD A, r	A ← A + r	↕	↕	V	↕	0	↕	10 [000] r	1	1	4	r Reg. 000 B 001 C 010 D 011 E 100 H 101 L 111 A
ADD A, n	A ← A + n	↕	↕	V	↕	0	↕	11 [000] 110 ← n →	2	2	7	
ADD A, (HL)	A ← A + (HL)	↕	↕	V	↕	0	↕	10 [000] 110	1	2	7	
ADD A, (IX+d)	A←A + (IX+d)	↕	↕	V	↕	0	↕	11 011 101 10 [000] 110 ← d →	3	5	19	
ADD A, (IY+d)	A←A+(IY+d)	↕	↕	V	↕	0	↕	11 111 101 10 [000] 110 ← d →	3	5	19	
ADC A, s	A ← A + s + CY	↕	↕	V	↕	0	↕	[001]				s is any of r, n, (HL), (IX+d), (IY+d) as shown for ADD instruction
SUB s	A ← A − s	↕	↕	V	↕	1	↕	[010]				
SBC A, s	A ← A − s − CY	↕	↕	V	↕	1	↕	[011]				
AND s	A ← A ∧ s	0	↕	P	↕	0	1	[100]				The indicated bits replace the 000 in the ADD set above.
OR s	A ← A ∨ s	0	↕	P	↕	0	0	[110]				
XOR s	A ← A ⊕ s	0	↕	P	↕	0	0	[101]				
CP s	A − s	↕	↕	V	↕	1	↕	[111]				
INC r	r ← r + 1	•	↕	V	↕	0	↕	00 r [100]	1	1	4	
INC (HL)	(HL) ← (HL)+1	•	↕	V	↕	0	↕	00 110 [100]	1	3	11	
INC (IX+d)	(IX+d) ← (IX+d)+1	•	↕	V	↕	0	↕	11 011 101 00 110 [100] ← d →	3	6	23	
INC (IY+d)	(IY+d) ← (IY+d) + 1	•	↕	V	↕	0	↕	11 111 101 00 110 [100] ← d →	3	6	23	
DEC m	m ← m−1	•	↕	V	↕	1	↕	[101]				m is any of r, (HL), (IX+d), (IY+d) as shown for INC. Same format and states as INC. Replace 100 with 101 in OP code.

Notes: The V symbol in the P/V flag column indicates that the P/V flag contains the overflow of the result of the operation. Similarly the P symbol indicates parity. V = 1 means overflow, V = 0 means not overflow. P = 1 means parity of the result is even, P = 0 means parity of the result is odd.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown.
↕ = flag is affected according to the result of the operation.

4.6.4 16-bits reken- en telinstructies

16-bits rekeninstructies zijn in principe gelijk aan 8-bits instructies. Er zijn er minder: alleen ADD, ADC en SUB zijn voor een aantal registerparen beschikbaar. Het resultaat van een operatie wordt in registerpaar HL opgeslagen (en niet in de accu, zoals bij 8-bits instructies). Bij ADD kunt u de resultaten ook in het indexregister opslaan. 16-bits instructies zijn in feite achter elkaar uitgevoerde 8-bits instructies. Omdat ze deze instructies automatisch verbinden, zijn ze sneller en korter.

```
16-bits        8-bits

ADD HL,BC       LD A,L
               ADD A,C
                LD L,A
                LD A,H
               ADC A,B
                LD H,A
```

Alle 16-bits instructies gebruiken impliciete adressering. De flag-beïnvloeding bij ADC en SBC is hetzelfde als bij de 8-bits instructies. ADD beïnvloedt alleen de carry en de 16-bits instructies INC en DEC negeren de flags.

assembler	BASIC
ADD IX,DE	IX=IX+DE
ADC HL,BC	HL=HL+BC+CF
SBC HL,SP	HL=HL-SP-CF

Twee voorbeelden:

ADD HL,DE

```
HL=&HC000
DE=&H0800

&HC000 = 1100 0000 0000 0000
&H0800 = 0000 1000 0000 0000 +
------   -------------------
&HC800 = 1100 1000 0000 0000
```

flags:

S	
Z	
V	
C	0

De S-, Z- en V-flag worden niet beïnvloed.

ADC HL,DE

```
HL=&HF800
DE=&H0800

 &HF800 =   1111 1000 0000 0000
 &H0800 =   0000 1000 0000 0000
-------     -------------------
&H10000 = 1 0000 0000 0000 0000
```

flags:

S	0
Z	0
V	1
C	1

Hier bevat HL niet het juiste resultaat &H10000, maar 0. De carry-flag geeft deze fout aan. Bij 16-bits operaties correspondeert de carry-flag met bit 16. De 16-bits telinstructies zijn allemaal impliciet geadresseerd en hebben betrekking op de 16-bits registers BC, DE, HL, SP, IX en IY. Ze beïnvloeden de flags niet, in tegenstelling tot 8-bits telinstructies.

Opgave

Schrijf een programmaatje om twee 8-bits getallen op te tellen. In BASIC slaat POKE de getallen op in het RAM. Het resultaat van de optelling komt ook in het RAM. Na de sprong terug naar BASIC leest de computer het resultaat met PEEK en voert het uit.

Oplossing

Omdat de 8-bits optelling de accu gebruikt, moet het eerste getal in de accu worden opgeslagen:

```
LD A,eerste getal
```

Het tweede getal slaat de computer op in een 8-bits register:

```
LD H,tweede getal
```

Nu voeren we de optelling uit:

```
ADD A,H
```

De som moet in &HF100 komen te staan:

```
LD (&HF100),A
```

Met &HF000 als beginadres krijgen we het volgende resultaat:

```
F000 3E10    10   LD   A,&H10
F002 2620    20   LD   H,&H20
F004 84      30   ADD  A,H
F005 3200F1  40   LD   (&HF100),A
F008 C9      50   RET
```

Uit de assemblerlisting blijkt dat het eerste getal (&H10) op adres &HF001 is opgeslagen en het tweede (&H20) op adres &HF003. Het BASIC-programma ziet er als volgt uit:

```
10 FOR I=&HF000 TO &HF008
20 READ A:POKE I,A:NEXT I
30 DEF USR1=&HF000
40 X=USR1(1)
50 PRINT PEEK(&HF100)
60 DATA &H3E,&H10,&H26,&H20,&H84,&H32,&H00,&HF1,&HC9
```

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments	
ADD HL, ss	HL ← HL+ss	↕	•	•	•	0	X	00 ss1 001	1	3	11	ss	Reg.
												00	BC
ADC HL, ss	HL←HL+ss+CY	↕	↕	V	↕	0	X	11 101 101	2	4	15	01	DE
								01 ss1 010				10	HL
												11	SP
SBC HL, ss	HL←HL-ss-CY	↕	↕	V	↕	1	X	11 101 101	2	4	15		
								01 ss0 010					
ADD IX, pp	IX ← IX + pp	↕	•	•	•	0	X	11 011 101	2	4	15	pp	Reg.
								00 pp1 001				00	BC
												01	DE
												10	IX
												11	SP
ADD IY, rr	IY←IY+rr	↕	•	•	•	0	X	11 111 101	2	4	15	rr	Reg.
								00 rr1 001				00	BC
												01	DE
												10	IY
												11	SP
INC ss	ss ← ss + 1	•	•	•	•	•	•	00 ss0 011	1	1	6		
INC IX	IX ← IX + 1	•	•	•	•	•	•	11 011 101	2	2	10		
								00 100 011					
INC IY	IY ← IY + 1	•	•	•	•	•	•	11 111 101	2	2	10		
								00 100 011					
DEC ss	ss ← ss - 1	•	•	•	•	•	•	00 ss1 011	1	1	6		
DEC IX	IX ← IX - 1	•	•	•	•	•	•	11 011 101	2	2	10		
								00 101 011					
DEC IY	IY ← IY - 1	•	•	•	•	•	•	11 111 101	2	2	10		
								00 101 011					

Notes: ss is any of the register pairs BC, DE, HL, SP
pp is any of the register pairs BC, DE, IX, SP
rr is any of the register pairs BC, DE, IY, SP.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

4.7 Sprongen

Sprongen zijn meestal voorwaardelijk, dat wil zeggen afhankelijk van de status van een flag. We gebruiken de flags H en N als we met BCD's rekenen (binary coded decimals). Deze flags kunnen niet worden getest. De andere flags (C, P/V, Z, S) kunnen bij een voorwaardelijke sprong worden getest.

carry flag (overdracht; afk.: C)

De carry flag (bit 0 van register F) heeft twee functies:

- hij geeft aan of bij optellen of aftrekken een overdracht plaatsvindt
- de instructies SRL, SRA, SLA, RR, RL, RRC, RLC, RRA, RLA, RRCA en RLCA gebruiken de carry als 9e bit

De BCD-roteerinstructies RLD, RRD beïnvloeden de carry niet. De logische instructies AND, OR en XOR zetten de carry steeds op 0. Ze kunnen dus worden gebruikt om de carry te wissen. De volgende instructies veranderen de carry:

NEG	carry flag wordt gezet als A voor de instructie 0 was
DAA	de beïnvloeding is hier gecompliceerd; omdat we de BCD-instructies niet hebben behandeld, gaan we er niet nader op in (zie paragraaf 5.4)
SCF	set carry flag; carry flag wordt gezet
CCF	complement carry flag; levert complement van carry

Alle andere instructies beïnvloeden de carry niet.

parity/overflow (pariteit/overloop; afk.: P/V)

De functies van deze flag zijn afhankelijk van de uitgevoerde instructie.

OVERFLOW: deze flag geeft een overflow aan bij de rekenoperaties ADD (8 bits), ADC, SUB, SBC, INC (8 bits), DEC (8 bits), NEG en bij CP. Dit betekent dat het voorteken van een getal foutief is veranderd. Uitzonderingen hierop zijn de 16-bits varianten van ADD, INC en DEC. Deze instructies beïnvloeden V niet.

PARITEIT: deze flag wordt gebruikt als P-flag voor inputinstructies (IN), roteer- en schuifinstructies RR, RL, RRC, RLC, RLD, RRD, SLA, SRA en SRL, de logische instructies AND, OR, XOR en DAA. P is 1 als het aantal gezette bits van een byte even is en 0 als dat oneven is. Uitzonderingen: RLA, RRA, RLCA, RRCA beïnvloeden P niet. Bij de blokinstructies LDD, LDI, CPD, CPI, CPDR en CPIR is P/V gereset als BC=0 (BC is het telregister). In het andere geval is P/V gezet. Daarom resetten LDDR en LDIR P/V altijd.

INTERRUPT FLAG: bij LD A,I en LD A,R krijgt P/V de waarde van de interrupt enable flipflops (IFF). Deze is 0 als de maskeerbare interrupts zijn geblokkeerd en 1 als ze zijn toegelaten. N.B.: de

BIT-instructie en alle blokin- en uitvoerinstructies zetten deze flag willekeurig, dat wil zeggen dat ze de vorige waarde veranderen. Andere instructies beïnvloeden deze flag niet.

zero flag (nul; afk.: Z)

De Z-flag geeft aan of de waarde van een byte nul is. Is dit het geval dan wordt Z gezet, zoniet gereset. Is er bij een vergelijkingsinstructie sprake van gelijkheid, dan wordt Z gezet en bij ongelijkheid gereset. De BIT-instructie zet de zero flag als de geteste bit 0 is, anders wordt deze flag gereset. De volgende instructies beïnvloeden de Z-flag:

rekeninstructies	ADD, ADC, SUB, SBC, INC, DEC, NEG, DAA
N.B.	de 16-bits instructies ADD, INC en DEC oefenen geen invloed uit op de zero flag
vergelijkings-instructies	CP: Z=1 bij gelijkheid, anders Z=0
bitinstructies	BIT
roteer-, schuif-instructies	RR, RL, RRC, RLC, SRL, SRA, SLA, RLD, RRD
N.B.	geen invloed: RRA, RLA, RRCA, RLCA
blokzoek-instructies	CPI, CPIR, CPD, CPDR: Z=1 bij gelijkheid
invoer-instructies	IN
laadinstructies	LD A,I of LD A,R

Blokin- en uitvoer: Z is gezet als na uitvoer B=0. INI, IND, OUTI, OUTD beïnvloeden Z op deze manier en INIR, INDR, OTIR, OTDR maken Z gelijk aan 1. Alle andere instructies beïnvloeden Z niet.

sign flag (voorteken; afk.: S)

De sign flag bevat de waarde van de voortekenbit, dat is bit 7. De volgende instructies beïnvloeden de S-flag:

reken- en logische instructies	ADD, ADC, SUB, SBC, INC, DEC, NEG, DAA, AND, OR, XOR, CP
roteer- en schuifinstructies	RL, RR, RLC, RRC, SRL, SRA, SLA, RLD, RRD
blokzoekinstructies	CPD, CPI, CPDR, CPIR
invoerinstructie	IN
laadinstructies	LD A,I en LD A,R

N.B.: ADD (16 bits), INC (16 bits), DEC (16 bits), RLA, RRA, RLCA, RRCA oefenen geen invloed uit op de sign flag.

De BIT-instructie en de blokin- en uitvoerinstructies INI, IND, OUTI, OUTD, INIR, INDR, OTIR, OTDR maken de S-flag willekeurig 0 of 1. Informatie over de manier waarop de instructies deze flag beïnvloeden, vindt u in de bijlage.

De Z80 kent vijf verschillende sprongen:

- sprongen binnen het hoofdprogramma (JUMP), GOTO in BASIC
- sprongen naar subroutines (CALL en RET), GOSUB en RETURN in BASIC
- relatieve sprongen (JUMP RELATIVE), te vergelijken met FOR-NEXT in BASIC
- restart instructies (RST), waarmee naar een van te voren bepaald vast adres wordt gesprongen; geen BASIC-equivalent
- interruptsprongen (zie stuurinstructies)

De eerste drie zijn bij de Z80 onvoorwaardelijk of voorwaardelijk, afhankelijk van de flagstatus. Bij voorwaardelijke sprongen kunnen de flags Z, C, P/V en S een sprong tot gevolg hebben. De status van elke flag kan worden getest. Voor de assembleertaal gelden de volgende afkortingen:

Z	spring indien nul	Z=1
NZ	spring indien niet nul	Z=0
C	spring indien overdracht	C=1
NC	spring indien geen overdracht	C=0
PO	spring indien oneven pariteit	P/V=0
PE	spring indien even pariteit	P/V=1
P	spring indien plus	S=0
M	spring indien min	S=1

Bovendien kent de Z80 een speciale lusinstructie die het B-register verlaagt en een relatieve sprong uitvoert zolang B niet 0 is. Deze instructie heet DJNZ (decrement (=verlaag) word register and jump if not zero).

4.7.1 JUMP

JP voert een sprong uit in het hoofdprogramma. De sprongadressen kunnen absoluut en voorwaardelijk worden geadresseerd:

```
JP nn
JP cc,nn
```

cc staat voor een voorwaarde (condition), dus voor Z, NZ, C, NC, PO, PE, P of M.

JP nn	springt onvoorwaardelijk naar het aangegeven adres
JP cc,nn	springt naar het aangegeven adres als aan de voorwaarde is voldaan. Als dat niet het geval is, gaat de procesor verder met de volgende instructie.

BASIC-equivalenten:

machinetaal	BASIC
JP nn	GOTO regelnummer
JP NZ,nn	IF Z=0 THEN GOTO regelnummer
JP Z,nn	IF Z=1 THEN GOTO regelnummer

De processor voert een sprong uit doordat hij het aangegeven adres in de PC leest. Vervolgens leest hij op deze plaats de volgende instructiecode en voert deze uit. Bij de absolute adressering volgt op de opcode van 1 byte het sprongadres in de volgorde lowbyte, highbyte. Omdat alle 3-bytes instructies tamelijk langzaam zijn, gebruiken we relatieve sprongen; dit zijn 2-bytes instructies. De indirect geadresseerde sprongen hebben een opcode van 1 byte. Indirecte adressering:

```
JP (X)
X: HL,IX of IY
```

JP (X) springt naar het adres in register X.

4.7.2 CALL/RET

We hebben al besproken hoe het bij CALL en RET zit met de terugsprongadressen in verband met de stack en wat de functie van de SP daarbij is. Een subroutine kan voorwaardelijk of onvoorwaardelijk worden opgeroepen.

```
CALL nn
CALL cc,nn
```

Het sprongadres (beginadres van de subroutine) wordt absoluut aangegeven. Bij de uitvoering vinden alle noodzakelijke stack-, SP- en PC-operaties plaats. Dat gaat als volgt: nadat de instructie helemaal is gelezen, wijst PC op de volgende instructie. Daarna komen de operaties.

```
SP-1=PC-highbyte
SP-2=PC-lowbyte
SP=SP-2
PC=nn
```

De volgende instructie staat in het adres waar PC op wijst. RET sluit een subroutine af. Ook RETURN kan onvoorwaardelijk of voorwaardelijk zijn.

```
RET
RET cc
```

Bij de uitvoering van RET gebeurt het volgende:

```
PC-lowbyte=SP
PC-highbyte=SP+1
SP=SP+2
```

Het programma gaat verder bij het adres dat van de stack is gehaald. RET is in tegenstelling tot CALL slechts 1 byte lang. Bij CALL geeft u het 16-bits adres aan, dat wil zeggen dat deze instructie 3 bytes telt. Er zijn twee speciale terugspronginstructies, RETI en RETN. Ze komen aan de orde in de paragraaf over stuurinstructies.

4.7.3 Restart

Dit type spronginstructie is minimaal 1 byte lang en is daarmee de snelste spronginstructie (op RET na). RST zorgt voor een subroutinesprong naar een adres in het onderste deel van het geheugen. Er zijn 8 RESTART-instructies. De sprongadressen van de restarts zijn: &H0, &H8, &H10, &H18, &H20, &H28, &H30 en &H38. Formaat:

```
RST p

p: een van de genoemde 8-bits adressen
```

Omdat RST de snelste spronginstructie is, staan in het onderste deel van het geheugen (&H0-&H40) zeer belangrijke, veelgebruikte routines of sprongen naar deze routines. De functies van de RST-instructies komen later een voor een ter sprake.

4.7.4 Jump relative

Om deze instructie te illustreren kijken we naar het programma uit paragraaf 1.2. In de assemblerlisting wordt de relatieve sprong op &HF00C op een andere manier opgevoerd dan in de corresponderende machinetaalinstructies. Die laatste bevatten de afstand (dis) die moet worden overbrugd ten opzichte van het actuele adres (inhoud van register PC), terwijl in de assemblerlisting gewoon de offset staat (het adres dat het doel is van de sprong). In machinetaal is de eerste byte de opcode en de tweede de afstand (twee-complementaire weergave met voorteken). Het formaat van de assemblerinstructie:

```
JR e
JR x,e

e  offset
x  Z,NZ,C,NZ
```

U kunt alleen maar relatief springen op grond van de C- en de Z-flag. Laten we de relatieve sprong op &HF00C uit paragraaf 1.2

eens goed analyseren. Het doel van de sprong is LD A,&H2A op adres &HF003. Het verschil is dus 9 (&HF00C-HF003). Omdat het om een terugwaartse sprong gaat (het doeladres is kleiner dan het adres van de spronginstructie) is de afstand -9. Daar moet 2 van worden afgetrokken. De reden hiervoor is dat de processor de hele instructie leest, in dit geval de opcode (byte 1) en de offset (byte 2). Daarna staat de PC op het beginadres van de volgende instructie, 2 hoger dan het adres van de spronginstructie en 5 hoger dan &HF003, het doel van de sprong. De Z80 voert, zoals gezegd, de sprong uit door de afstand bij het adres in PC (het actuele adres) op te tellen. De verhoging van PC met 2 ten opzichte van het sprongadres mag niet worden genegeerd: bij een terugwaartse sprong moet het programma ook over deze twee bytes heenspringen. De afstand tot de offset (&HF003) is dus:

-9-2=-11 &HF5 als twee-complement

Deze byte staat in het machinetaalprogramma op adres &HF00D, in de opcode na &HF00C. In plaats van &HF003 had er na de instructie JR ook &HF00C-3 kunnen staan. Een dergelijk verschil zou betekenen dat er met een ander assembleerprogramma werd gewerkt. Hoewel in assemblerinstructies 16-bits adressen staan, gaat het wel degelijk om relatieve sprongen. Rekening houdend met de waarde die moet worden afgetrokken, zijn relatieve sprongen mogelijk van +129 tot en met -126 bytes.

Opgave

De belangrijkste instructies hebben we gehad. Met relatieve sprongen kunnen we nu lussen programmeren. Subroutines aanroepen is heel belangrijk. Naast zelf geprogammeerde subroutines beschikt u ook over routines die al in het ROM aanwezig zijn, de systeemroutines (zie hoofdstuk 6). Voor we zover zijn komen we al een paar nuttige systeemroutines tegen. VPOKE uit het demo-programma (paragraaf 1.2) kent u al. In feite komt deze overeen met de instructie LD (HL),A, hoewel de gegevens niet worden opgeslagen in het gewone RAM maar in het video-RAM. Systeemroutines eindigen met RET zodat na de gewenste actie ons eigen programma verder gaat. Het startadres van VPOKE is &H7CD. Schrijf met behulp van deze routine een programma dat de hele karakterset (ASCII-codes 0-255) in SCREEN 0 achter elkaar op het scherm zet. Het video-RAM loopt van adres &H0 tot &H3C0. Schrijf

het programma eventueel eerst in BASIC. Let bij de omzetting in machinetaal goed op de berekening van de offset van de lus.

Oplossing

Eerst de FOR/NEXT-lus in BASIC:

```
10 FOR I=0 TO 255
20 VPOKE I,I
30 NEXT
```

De assemblerlisting van het machinetaalprogramma is:

```
F000 3E00      10    LD   A,0
F002 210000    20    LD   HL,0
F005 CDCD07    30    CALL &H7CD
F008 23        40    INC  HL
F009 3C        50    INC  A
F00A 20F9      60    JR   NZ,&HF005
F00C C9        70    RET
```

Verklaring:

10	zet de waarde in de accu die moet worden weggeschreven op 0
20	slaat een 0 op in adresregister HL
30	het begin van de lus; oproep van de systeemroutine die de waarde in de accu wegschrijft naar adres HL van het video-RAM
40	de waarde (A) wordt met 1 vermeerderd
50	het adres (HL) wordt met 1 vermeerderd
60	JR, de spronginstructie, bepaalt of het programma klaar is of niet. JR verhoogt de accu direct en beïnvloedt in overeenstemming daarmee de flags. De test is dus: "Is de accu nog ongelijk aan 0 na verhoging?" Het resultaat blijft waar tot de accu bij het begin van de lus de waarde 255 heeft. Bij de volgende INC-instructie moet dat eigenlijk 256 worden. Aangezien de accu alleen 1-bytes getallen verwerkt, wordt de waarde 0 (255+1=00+carry=1). De lus wordt dan niet meer herhaald en RET beëindigt het programma.

Mnemonic	Symbolic Operation	C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
JP nn	PC ← nn	•	•	•	•	•	•	11 000 011	3	3	10	
								← n →				
								← n →				
JP cc, nn	If condition cc is true PC ← nn, otherwise continue	•	•	•	•	•	•	11 cc 010	3	3	10	
								← n →				
								← n →				
JR e	PC ← PC + e	•	•	•	•	•	•	00 011 000	2	3	12	
								← e−2 →				
JR C, e	If C = 0, continue	•	•	•	•	•	•	00 111 000	2	2	7	If condition not met
								← e−2 →				
	If C = 1, PC ← PC+e								2	3	12	If condition is met
JR NC, e	If C = 1, continue	•	•	•	•	•	•	00 110 000	2	2	7	If condition not met
								← e−2 →				
	If C = 0, PC ← PC + e								2	3	12	If condition is met
JR Z, e	If Z = 0 continue	•	•	•	•	•	•	00 101 000	2	2	7	If condition not met
								← e−2 →				
	If Z = 1, PC ← PC + e								2	3	12	If condition is met
JR NZ, e	If Z = 1, continue	•	•	•	•	•	•	00 100 000	2	2	7	If condition not mc
								← e−2 →				
	If Z = 0, PC ← PC + e								2	3	12	If condition met
JP (HL)	PC ← HL	•	•	•	•	•	•	11 101 001	1	1	4	
JP (IX)	PC ← IX	•	•	•	•	•	•	11 011 101	2	2	8	
								11 101 001				
JP (IY)	PC ← IY	•	•	•	•	•	•	11 111 101	2	2	8	
								11 101 001				
DJNZ,e	B ← B-1 If B = 0, continue	•	•	•	•	•	•	00 010 000	2	2	8	If B = 0
								← e−2 →				
	If B ≠ 0, PC ← PC + e								2	3	13	IF B ≠ 0

cc	Condition
000	NZ non zero
001	Z zero
010	NC non carry
011	C carry
100	PO parity odd
101	PE parity even
110	P sign positive
111	M sign negative

Notes: e represents the extension in the relative addressing mode.

e is a signed two's complement number in the range <−126, 129>

e−2 in the op-code provides an effective address of pc +e as PC is incremented by 2 prior to the addition of e.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

Mnemonic	Symbolic Operation	C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
CALL nn	$(SP-1) \leftarrow PC_H$ $(SP-2) \leftarrow PC_L$ $PC \leftarrow nn$	•	•	•	•	•	•	11 001 101 ← n → ← n →	3	5	17	
CALL cc, nn	If condition cc is false continue, otherwise same as CALL nn	•	•	•	•	•	•	11 cc 100 ← n → ← n →	3 3	3 5	10 17	If cc is false If cc is true
RET	$PC_L \leftarrow (SP)$ $PC_H \leftarrow (SP+1)$	•	•	•	•	•	•	11 001 001	1	3	10	
RET cc	If condition cc is false continue, otherwise same as RET	•	•	•	•	•	•	11 cc 000	1 1	1 3	5 11	If cc is false If cc is true
RETI	Return from interrupt	•	•	•	•	•	•	11 101 101 01 001 101	2	4	14	
RETN	Return from non maskable interrupt	•	•	•	•	•	•	11 101 101 01 000 101	2	4	14	
RST p	$(SP-1) \leftarrow PC_H$ $(SP-2) \leftarrow PC_L$ $PC_H \leftarrow 0$ $PC_L \leftarrow P$	•	•	•	•	•	•	11 t 111	1	3	11	

cc	Condition	
000	NZ	non zero
001	Z	zero
010	NC	non carry
011	C	carry
100	PO	parity odd
101	PE	parity even
110	P	sign positive
111	M	sign negative

t	P
000	00H
001	08H
010	10H
011	18H
100	20H
101	28H
110	30H
111	38H

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown
↕ = flag is affected according to the result of the operation.

4.8 Logische instructies

Logische instructies behoren tot de instructies voor dataverwerking. De Z80 kent de logische instructies AND, OR, XOR (exclusive OR) en de vergelijkingsinstructie CP. Deze instructies werken met gegevens van 8 bits. De accu is het register dat de logische operatie uitvoert en wordt daarom niet genoemd in de operand van de assemblerinstructie, in tegenstelling tot bijvoorbeeld ADD A,B. De vier instructies AND, OR, XOR en CP komen voor met de volgende adresseringen:

- impliciet: register A, B, C, D, E, H, L
- indirect: register (HL)
- geïndiceerd
- onmiddellijk

Logische instructies leggen logische verbanden. Het resultaat van logische verbindingen is waar als de afzonderlijke uitspraken waar zijn. In de formele logica betekent het cijfer 1 dat een bewering waar is en 0 dat de bewering onwaar is. Deze binaire conventie kan voor computerdoeleinden onverkort gehandhaafd blijven. Met de logische operator AND zijn er de volgende mogelijkheden:

```
1 AND 1 = 1  beide uitspraken waar    --> resultaat waar
1 AND 0 = 0  een uitspraak onwaar     --> resultaat onwaar
0 AND 1 = 0  een uitspraak onwaar     --> resultaat onwaar
0 AND 0 = 0  beide uitspraken onwaar  --> resultaat onwaar
```

De logische operator AND is een instructie in MSX-BASIC. Probeer eens:

```
PRINT 1 AND 1
PRINT 1 AND 0 enzovoort
```

Een 1 betekent in de hardware dat er een stroompje loopt en een 0 dat dat niet het geval is. De logische operaties vinden als volgt plaats: twee ingangslijnen zijn aangesloten op een circuit met een uitgangslijn. Afhankelijk van de toestand van de invoerlijnen (stroom of geen stroom) bepaalt het circuit of de uitgangslijn actief is of niet. De schakelingen in de microprocessor zijn in algebraïsche termen te vatten (Boole algebra). Een microprocessor bestaat uit een aantal op elkaar aangesloten logische gates. Voor een eenvoudige optelling zijn al verschillende logische operaties nodig. Bij het programmeren hebben we met de hardware-structuren niets te maken. In programma's hebben de logische operaties

betrekking op 8 bits. De bits van de twee bytes worden steeds met elkaar verbonden:

```
11111000  AND
01010011
--------
01010000

bit 0  0 AND 1 = 0
bit 1  0 AND 1 = 0
bit 2  0 AND 0 = 0
bit 3  1 AND 0 = 0
bit 4  1 AND 1 = 1
bit 5  1 AND 0 = 0
bit 6  1 AND 1 = 1
bit 7  1 AND 0 = 0
```

Een belangrijke toepassing van AND is het wissen of buiten beschouwing laten van bepaalde bits.

```
A=&B10111001
```

Stel dat we alleen de bits 0-3 willen bekijken, dan doen we dat op de volgende manier:

```
10111001  AND
00001111  masker
--------
00001001
```

Het tweede getal zorgt ervoor dat de bits 4-7 van het eerste getal buiten beschouwing worden gelaten. Daartoe zijn de bits 0-3 gezet en de bits 4-7 niet. In BASIC geformuleerd:

```
A=&B10111001
A=A AND &B00001111
```

In machinetaal:

```
LD A,&B10111001
AND  &B00001111
```

Bij OR moet minstens 1 van de uitspraken waar zijn, wil het resultaat waar zijn:

```
0 OR 0 = 0
0 OR 1 = 1
1 OR 0 = 1
1 OR 1 = 1
```

Met OR kunt u verschillende bits van een byte zetten.

```
A=&B10001011
```

De hoogste 3 bits (5, 6 en 7) zet u op de volgende manier:

```
10001011  OR
11100000  masker
--------
11101011
```

Het tweede getal heeft een 1 voor bits die moeten worden gezet en een 0 voor bits die niet moeten veranderen.

```
assembler            BASIC

LD A,&B10001011      A=&B10001011
OR &B11100000        A=A OR &B11100000
```

Bij XOR (exclusive OR) is het resultaat alleen waar als slechts een van beide beweringen waar is. Vandaar de term exclusive: de beweringen sluiten elkaar uit. Zijn ze beide waar of beide onwaar, dan is het resultaat 0:

```
0 XOR 0 = 0
0 XOR 1 = 1
1 XOR 0 = 1
1 XOR 1 = 0
```

Vergelijken of een complement berekenen doet u door de bytes in kwestie te verbinden met XOR. Het resultaat is 0 bij gelijke bytes. Als de bytes niet gelijk zijn, zijn in het resultaat de bits gezet die van elkaar verschillen. Twee gevallen waarin de bytes worden vergeleken:

```
10101010
10101010  XOR
--------
00000000


10101010
10101100  XOR
--------
00000110  alleen bit 1 en 2 verschillen
```

Een complement berekenen gebeurt ook met een tweede getal (masker). Een 1 betekent dat u het complement wilt hebben en een 0

dat de bit gelijk blijft. In het volgende voorbeeld gaat het dus om het complement van bit 4-7.

```
10101111
11110000  XOR
--------
01011111
```

BASIC-equivalenten:

machinetaal	BASIC
AND H	A=A AND H
OR (HL)	A=A OR PEEK(HL)
XOR &HFF	A=A XOR &HFF

Bij logische instructies zet de computer de carry altijd op 0. De Z- en S-flag worden op de normale wijze beïnvloed. De P/V-flag geeft bij deze instructies de pariteit aan van het resultaat. De pariteit is 1 als het aantal enen in een byte even is en 0 als dit aantal oneven is.

Opgaven

(1) Wat is het resultaat van:

OR met &HFF ?
OR met &HO ?
AND met &HFF ?
AND met &HO ?
XOR met &HFF ?
XOR met &HO ?

(2) Vertaal de BASIC-instructie NOT op twee manieren in machinetaal (met betrekking tot de accu).

Oplossingen

```
(1) OR  &HFF  -->  &HFF: alle bits zijn gezet
    OR  &HO   -->  geen verandering
    AND &HFF  -->  geen verandering
    AND &HO   -->  &HO: alle bits zijn gereset
    XOR &HFF  -->  van alle bits is het complement gevormd
    XOR &HO   -->  geen verandering
```

```
(2) XOR-instructie: XOR &HFF
    CPL-instructie: CPL
```

4.8.1 De vergelijkingsinstructie CP

We gaan verder met de vergelijkingsinstructie CP (compare). CP vergelijkt de accu-inhoud met een byte. Deze kan als volgt worden geadresseerd:

- impliciet: register A, B, C, D, E, H, L
- indirect: registerpaar (HL)
- geïndiceerd
- onmiddellijk

CP trekt de geadresseerde byte van de accu af en beïnvloedt de flags, afhankelijk van het resultaat. In tegenstelling tot SUB wordt het resultaat niet in de accu opgeslagen. De instructie beïnvloedt de accu-inhoud dus niet. Afhankelijk van de status van de flags voert de computer na deze instructie een voorwaardelijke sprong uit. Bij een vergelijking zijn er drie mogelijkheden:

accu-inhoud groter

De carry is in dit geval steeds 0, want het resultaat kan niet groter zijn dan 255.

accu-inhoud gelijk

In dit geval is Z=1, want het resultaat van de aftrekking is 0. Tevens is C=0 omdat er geen overdracht plaatsvindt.

accu-inhoud kleiner

Omdat er een negatieve overdracht plaatsvindt, is de carry-flag hier steeds gezet.

Regels:

```
C=0  betekent  >=
Z=1  betekent  =
C=1  betekent  <
```

verder geldt:

```
Z=0           betekent  <>
C=0 en Z=0    betekent  >
C=1 of Z=0    betekent  =<
```

Deze regels gelden alleen voor bytes tussen 0-255 zonder voorteken. Voor bytes met voorteken, die twee-complementaire getallen voorstellen, gelden ingewikkelder regels. Deze leidt u af uit de flag-regels voor berekeningen met voortekens. De Z-flag geeft aan of er sprake is van gelijkheid. Groter of kleiner dan is af te lezen aan de status van de S- of V-flag. XOR verbindt de S- met de V-flag: als V is gezet (overflow), vormt de computer het complement van S. In het andere geval blijft de toestand van S gehandhaafd.

```
S XOR V = 0  betekent  >=
S XOR V = 1  betekent  <
```

Vanaf hier vermelden we niet meer dat de bytes als getallen zonder voorteken dienen te worden geïnterpreteerd.

CP B

```
A=&H35=00110101
B=&H21=00100001  geen twee-complement

00110101
00100001 -
--------
00010100
```

flags:

```
S    0    bit=0
Z    0    <> 0
V    0    geen overflow
C    0    geen overdracht
```

De carry-flag is gelijk aan 0. Hieruit volgt dat de accu-inhoud groter is dan die van de vergelijkbare byte (inhoud van het B-register).

CP C

```
A=&H01=00000001
C=&H81=10000001

 00000001
 10000001 -
 ---------
110000000
```

flags:

```
S    1    bit 7=1
Z    0    <>
V    1    geen overdracht van 6 naar 7, maar
C    1    overdracht van 7 naar 8
```

C is dus 1. Hieruit kunt u opmaken dat de vergelijkingswaarde (de inhoud van het C-register) groter is dan de accu-inhoud. We gebruiken de CP-instructie vaak in combinatie met test- of spronginstructies. De logische instructies staan in de lijst met 8-bits rekenkundige instructies (zie paragraaf 4.6).

Opgave

Met de logische instructies kunnen we bijvoorbeeld hoofdletters in kleine letters veranderen en omgekeerd. Vergelijk de ASCII-codes van de letters a en z:

```
A    &H41=&B01000001
a    &H61=&B01100001
```

```
Z    &H5A=&B01011010
z    &H7A=&B01111010
```

Bij kleine letters is bit 5 altijd 1 en bij hoofdletters altijd 0. Schrijf een programma dat alle hoofdletters in kleine letters verandert. Gebruik om in het video-RAM te schrijven de routine uit de vorige paragraaf (&H7CD). Op adres &H7D7 staat de routine die het video-RAM leest. Deze vervangt de instructie LD A,(HL) waarbij HL staat voor een adres in het video-RAM.

Oplossing

```
F000 210000     10          LD   HL,00
F003 CDD707     20  LUS     CALL &H07D7
F006 FE5C       30          CP   92
F008 30FE       40          JR   NC,GROTER
F00A FE41       50          CP   65
F00C 38FE       60          JR   C,GROTER
F00E F620       70          OR   &B00100000
F010 CDCD07     80          CALL &H07CD
F013 23         90  GROTER  INC  HL
F014 3E04      100          LD   A,4
F016 BC        110          CP   H
F017 20EA      120          JR   NZ,LUS
F019 C9        130          RET
```

Toelichting:

10	adreswijzer HL op 0
20	inhoud van video-RAM op adres HL in accu laden
30	is A>=ASC("Z")+1 ?
40	zo ja, dan niet omzetten (groter)
50	is A<ASC("A") ?
60	zo ja, dan niet omzetten (groter)
70	bit 5 zetten
80	nieuwe waarde in video-RAM schrijven
90	adres verhogen
100	highbyte naar accu ter vergelijking
110	is de highbyte van het adres >=4 ?
120	zoniet, dan opnieuw beginnen (lus)
130	zo ja: einde

Wilt u alle kleine letters in hoofdletters veranderen, dan is het voldoende in regel 30 en 50 de CP-instructies te wijzigen in respectievelijk &H7B en &H61. Bovendien vervangt u OR &B00100000 door AND &B11011111. Ook het bedrijfssysteem zet kleine letters om in hoofdletters. Als u BASIC-programma's intypt, kunt u naar keuze kleine letters of hoofdletters gebruiken; intern wordt alles in hoofdletters afgehandeld.

4.9 De assembler

Wie over een assembler beschikt, is de koning te rijk. Stuk voor stuk machinetaalinstructies vertalen, opcodes opzoeken, enzovoort - het behoort allemaal tot het verleden. Een assembler produceert de hexadecimale machinetaalinstructies die corresponderen met een in assembleertaal geschreven programma. Daarbij worden bijvoorbeeld sprongafstanden automatisch berekend. Assemblerprogramma's voor de Z80 zijn gebonden aan bepaalde regels. Het formaat van een assemblerregel ziet er als volgt uit:

```
label  instructie  operand ; commentaar
```

Om het programma in te voeren gebruikt u de BASIC-editor. Daarom heeft elke assemblerinstructie een regelnummer. We definiëren nu het invoerformaat van de assembler. Het juiste formaat voorkomt onnodige fouten. Werk dit deel daarom grondig door.

label

Aan het begin van een regel kan een label staan. Een label is een variabele. De lengte van de variabelenaam (labelnaam) mag niet groter zijn dan 6 karakters. Labelnamen moeten met een letter beginnen. Assemblerinstructies mag u niet als labelnaam gebruiken. Met labels kunnen sprongen makkelijker worden geprogrammeerd.

```
BEG  instructie ;BEG is een label
        ...
        ...
     JR BEG       ;spring naar BEG
        ...
        ...
```

Zoals gezegd berekent de assembler automatisch de juiste afstand.

instructie (mnemonic)

Na het eventueel aanwezige label volgt, na een spatie, de instructie. De mnemonic moet een geldige assemblerinstructie zijn, bijvoorbeeld LD, ADD, INC.

operand

Na de instructie en weer een spatie volgt de operand. Bij sprongen kunt u het sprongadres als label gebruiken (zie boven). In plaats van constanten of afstanden kunt u variabelenamen of labels gebruiken. In een operand mogen nooit spaties voorkomen.

commentaar

In onze assembler moet het commentaar bij een regel gescheiden zijn door een spatie en een puntkomma, anders krijgt u een syntax error. Door die scheiding weet de machine dat het commentaar geen deel uitmaakt van het programma.

Tijdens de vertaling produceert de assembler een listing die op printer of beeldscherm kan worden uitgevoerd. Bovendien kunt u de geproduceerde code opnemen op cassette of schijf. De assemblerlisting heeft de volgende structuur:

hex. adres	hex. code	BASIC-regel	label	in-struc-tie	operand	; commentaar
F003	36CC	50	verder	LD	(hl),bitmat	; bitmatrix
F005	23	60		INC	hl	; verhoog hl

Naast de Z80-instructies kent de assembler een serie pseudo-instructies. Dit zijn aanwijzingen voor de assembler, zoals END. END betekent dat er niet meer naar andere instructies hoeft te worden gezocht en dat de vertaling kan worden beëindigd. Een andere belangrijke instructie is EQU (equal: gelijk). Met EQU definiëert u de waarde van een variabele.

variabelenaam EQU waarde .

De instructie ORG (organisatie) geeft aan vanaf welk adres het

programma moet worden opgeslagen. Het beginadres is meestal &HF000. Zoals we al eerder hebben opgemerkt, dient u de volgende conventies in acht te nemen bij hexadecimale en binaire getallen:

- aan hexadecimale getallen behoort &H vooraf te gaan
- aan binaire getallen behoort &B vooraf te gaan
- aan octale getallen behoort &O vooraf te gaan

Zonder deze tekens begrijpt de computer uw bedoeling niet en interpreteert hij de tekens als decimale getallen. De standaardafspraken voor de Z80-assembler zijn een H na een hexadecimaal getal en een B na een binair getal. Wij wijken daar dus van af. Test de assembler door een klein programma in te voeren. Het source-programma in assembler kan onafhankelijk van het assemblerprogramma worden ingevoerd. Het eerste programma uit paragraaf 1.2 ziet er dan zo uit:

```
 10 ' ORG &HF000
 20 ' LD HL,0              ;start beeldschermgeheugen
 30 ' VERDER LD A,&H2A     ;= 42 = ASCII-code van *
 40 ' CALL &H07CD          ;komt overeen met VPOKE
 50 ' INC HL               ;verhoogt adres
 60 ' LD A,4
 70 ' CP H                 ;H>4?
 80 ' JR NZ,VERDER         ;nee, dan nog eens
 90 ' RET                  ;terug naar BASIC
100 ' END
```

Bij de invoer kunt u zowel kleine letters als hoofdletters gebruiken. Let erop dat u na elk regelnummer met 1 spatie tussenruimte ' invoert. Vergeet u dit, dan kan de assembler de regel niet vertalen en verschijnt de foutmelding

De ' ontbreekt in regel ...

Regel 10 bepaalt vanaf welk adres het programma wordt opgeslagen. In de programmalus gebruikt u het label VERDER als doel van de sprong. De rest van het programma bestaat uit normale assemblerinstructies. Een commentaar in een regel volgt na een puntkomma, voorafgegaan door een spatie. Spaties verdelen de regel in eenheden die de assembler kan herkennen. Daarom moeten tussen label, instructie, operand en commentaar altijd spaties staan en mogen bijvoorbeeld in een operand nooit spaties worden gebruikt. (HL) is dus fout, het moet (HL) zijn. Sla het ingevoerde programma op met SAVE"CAS:naam" (aanhalingstekens zijn absoluut noodzakelijk) en laad daarna de assembler met MERGE. De assembler neemt de regelnummers vanaf 10000 en regel 1 in beslag. Deze zijn dus niet beschikbaar voor het source-programma. Omdat

het langer duurt om zo'n bestand te laden, moet u altijd eerst de assembler (als normaal BASIC-programma) laden en daarna het source-programma met MERGE"CAS:naam". Start het programma met RUN. De assembler vraagt naar de naam van het programma, vraagt of er een assemblerlisting moet komen en of die moet worden geprint. De default antwoorden (j bij de listing en n bij de printer) kiest u met RETURN. Daarna begint de eigenlijke vertaling. De assemblerlisting komt op het scherm te staan. Foutmeldingen voert de computer uit voorafgaand aan de betreffende regel. Aan het einde van de listing wordt aangegeven of er ongedefinieerde labels en variabelen zijn. Daarna volgen programmanaam, beginadres, eindadres, programmalengte en aantal fouten. Fouten kunt u in de BASIC-regels corrigeren. Aan het einde van de listing verschijnt een tabel van alle labels en variabelen met hun waarden in de volgorde waarin ze voorkomen. Tenslotte vraagt het programma of de machinetaalcode moet worden opgeslagen. Als u j antwoordt, slaat het programma de code op als binair bestand onder de ingevoerde naam. Na assemblage staat het machinetaalprogramma op de aangegeven plaats in het geheugen. U kunt het met functietoets 1 oproepen. Op het startadres is met DEFUSR1 de instructie USR1 komen te staan. Bovendien bevat functietoets 1 nu de instructiereeks X=USR1(1)+RETURN. Als voorbeeld volgt hier de complete assemblerlisting van het programma dat we als voorbeeld hebben genomen.

```
F000          10          ORG  &HF000
F000 210000   20          LD   HL,0          ;start beeldschermgeh.
F003 3E2A     30   VERDER LD   A,&H2A        ;=42=ASCII-code voor *
F005 CDCD07   40          CALL &H07CD        ;komt overeen met VPOKE
F008 23       50          INC  HL            ;adres verhogen
F009 3E04     60          LD   A,4
F00B BC       70          CP   H             ;H>4?
F00C 20F5     80          JR   NZ,VERDER     ;nee, dan nog eens
F00E C9       90          RET                ;terug naar BASIC

programma: DEMO
start: &HF000   einde: &HF00E
lengte: &HF
fouten: 0
variabeletabel:
VERDER F003
opslaan (j/n)?
```

Probeert u bij het intypen van de listing de structuur van de assembler te doorgronden. De erop volgende toelichting kan daarbij behulpzaam zijn. Om de invoer van lange source-programma's te vergemakkelijken hebben we een programma geschreven waarin de AUTO-instructie zo is gedefinieerd dat achter

elk regelnummer een ' komt te staan.

```
  10 REM AUTO met toevoeging van '
  20 CLEAR 200,&HEFFF
  30 POKE &HFF0D,&H20
  40 POKE &HFF0E,&HF3
  50 DEFINT I
  60 DI=&HF320-&H4137
  70 FOR I=&H4137 TO &H415E:POKE I+DI,PEEK(I):NEXT
  80 POKE &H4148+DI,&H16+6
  90 I=I+DI
 100 READ A$:IF A$<>"#" THEN POKE I,VAL("&H"+A$):I=I+1:
     GOTO 100
 110 DATA 3E,27,DF,3E,20,DF,E1,C3,5F,41
 120 DATA #
 130 REM Inschakelen met
 140 REM POKE &HFF0C,&HC3
 150 REM Uitschakelen met POKE &HFF0C,&HC9
```

Probeer de in regel 130 en 140 gegeven mogelijkheden uit. Dit toegevoegde programma bevindt zich op adres &HF320 e.v. zodat dit gebied niet meer beschikbaar is voor eigen machinetaalprogramma's. Wilt u de AUTO-instructie weer op de normale manier gebruiken, voer dan POKE &HFF0C,&HC9 in.

4.9.1 De listing van de assembler

```
    1 CLEAR 200,&HEFFF:MAXFILES=0:GOTO10000
10000 REM ********* Z80 assembler *********
10010 GOSUB 13720
10020 LOCATE 4,4:PRINT"Z 8 0  -  a s s e m b l e r"
10030 LOCATE 3,8:INPUT"programmanaam ";NA$
10040 LOCATE 19,11:PRINT"j";
10050 LOCATE 3,11:INPUT"listing (j/n) ";A$
10060 IF (ASC(A$)OR 32)=110 THEN LF=0:GOTO 10100 ELSE LF=-1
10070 LOCATE 19,13:PRINT"n";
10080 LOCATE 3,13:INPUT"printer (j/n) ";A$
10090 IF (ASC(A$)OR 32)=106 THEN POKE &HFEE4,&HC3
10100 CLS:REM start assemblage ----------
10110 NR=FNDK(BP):BP=BP+2
10120 RN=FNDK(BP)
10130 IF RN>9999 THEN PRINT"END ontbreekt":GOTO 10590
```

```
10140 BP=BP+2
10150 IF (FNDK(BP)<>&H8F3A) OR (PEEK(BP+2)<>&HE6) THEN PRINT
"De ' ontbreekt in regel";RN:BP=NR:FA=FA+1:GOTO 10110
10160 BP=BP+3
10170 POKE VP,NR-BP-1
10180 POKE VP+1,BP-INT(BP/256)*256
10190 POKE VP+2,INT(BP/256)-256*(BP<0)
10200 RA$=R$:BP=NR
10210 FOR I= 0 TO 3:A$(I)="":NEXT
10220 PO=INSTR(R$,";")
10230 IF PO=0 THEN OP$="":GOTO 10260
10240 OP$=RIGHT$(R$,LEN(R$)-PO+1)
10250 POKE VP,PO-1
10260 J=0:R$=USR9(R$)
10270 IF LEFT$(R$,1)=" " THEN R$=RIGHT$(R$,LEN(R$)-1):GOTO 10270
10280 PO=INSTR(R$," ")
10290 IF R$="" THEN J=J-1:GOTO 10360
10300 IF PO=0 THEN 10350
10310 A$(J)=(LEFT$(R$,PO-1)):R$=RIGHT$(R$,LEN(R$)-PO)
10320 IF R$="" THEN J=J-1:GOTO 10360
10330 IF J>2 THEN ERROR 60
10340 J=J+1:GOTO 10270
10350 A$(J)=(R$)
10360 IF J>2 THEN ERROR 60
10370 GOSUB 10790
10380 REM uitvoer ----------
10390 IF FI THEN 10520
10400 IF NOT LF THEN LOCATE 5,3:PRINTRN:GOTO 10480
 ELSE IF POS(0)=0 THEN PRINT" ";ELSE
10410 PRINTRIGHT$("000"+HEX$(MP),4);" ";
10420 IF LP=0 THEN 10450
10430 FOR I=1 TO LP:PRINTRIGHT$("0"+HEX$(PW(I)-256*(PW(I)<0)),2);
10440 POKE MP+I-1,PW(I)-256*(PW(I)<0):NEXT I
10450 PRINTTAB(15);USING"####";RN;
10460 PRINTTAB(21);LB$;TAB(28);IN$;TAB(33);OU$;:IF OP$<>""
 THEN PRINT" ";
10470 PRINT OP$
10480 MP=MP+LP+DS:IF MP+4>&HF31F THEN PRINT:PRINT"Geheugen vol":
PRINT:GOTO 13090
10490 LP=0:DS=0
10500 LB$="":LA$="":IN$="":OU$="":OD$="":OP$=""
10510 GOTO 10110
10520 FOR J=LP TO 1 STEP -1
10530 PW(J+1)=PW(J):NEXT
10540 PW(1)=WI:LP=LP+1
10550 IF NOT DF THEN 10580
10560 IF LP=3 THEN PW(4)=PW(3)
10570 PW(3)=DW:LP=LP+1
```

```
10580 FI=0:DF=0:GOTO 10400
10590 REM einde programma ----------
10600 IF NOT LF THEN LOCATE 0,9
10610 PRINTSTRING$(36,""""):IF OP=0 THEN 10650
10620 FOR I=0 TO OP-1
10630 PRINT"ongedefinieerde ";OL$(I);" in ";OG(I,0);
" / adres &H";RIGHT$("000"+HEX$(OG(I,1)),4)
10640 FA=FA+1:NEXT
10650 PRINT:PRINT"programma: ";NA$
10660 PRINT"begin: &H";RIGHT$("000"+HEX$(MS),4);"   einde:
 &H";RIGHT$("000"+HEX$(MP-1),4)
10670 PRINT"lengte: &H";HEX$(MP-MS);" bytes"
10680 PRINT"fout: ";FA
10690 IF LT=0 THEN 10730
10700 PRINT"variabelen: "
10710 FOR I=0 TO LT-1
10720 PRINTLEFT$(LT$(I)+"       ",7);RIGHT$
("0000"+HEX$(WL(I)),4),:NEXT
10730 PRINT:POKE &HFEE4,&HC9
10740 LOCATE 15,CSRLIN:PRINT"n";
10750 LOCATE 0,CSRLIN:INPUT"opslaan (j/n)";A$
10760 IF (ASC(A$)OR 32)<>106 THEN 10780
10770 BSAVE "CAS:"+NA$+".BIN",MS,MP
10780 DEFUSR1=MS:POKE &HFEE4,&HC9:KEY 1,"X=USR1(1)"+CHR$(13):
KEY ON:END
10790 J=0:REM interpretatie ----------
10800 IN$=LEFT$(A$(J)+"    ",4)
10810 PO=INSTR(T1$,IN$)
10820 IF PO<>0 THEN LP=0:GOTO 11390
10830 PO=INSTR(T2$,IN$)
10840 IF PO<>0 THEN LP=1:GOTO 11080
10850 PO=INSTR(T3$,IN$)
10860 IF PO<>0 THEN LP=2:PW(1)=&HED:GOTO 11110
10870 PO=INSTR(T4$,IN$)
10880 IF PO<>0 THEN 11140
10890 IF J>0 THEN ERROR 60
10900 IF A$(0)="" THEN RETURN
10910 A$=A$(0):GOSUB 13210
10920 IF NO THEN ERROR 60
10930 LB$=LA$:WA=MP
10940 LT$(LT)=LA$:WL(LT)=MP:LT=LT+1
10950 FOR I=0 TO OP:IF LA$=OL$(I) THEN 10980 ELSE 10960
10960 NEXT
10970 J=J+1:GOTO 10800
10980 ON OG(I,2) GOTO 10990,11010
10990 SO=OG(I,1)-1:DA=WA:GOSUB 13660
11000 PW(1)=OS:GOTO 11030
11010 PW(2)=INT(WA/256)
```

```
11020 PW(1)=WA-PW(2)*256
11030 IF LF THEN PRINT"**** regel ";OG(I,0);" : ";OL$(I);
"=&H";HEX$(WA);:IF OG(I,2)=1 THEN PRINT" offset ";HEX$(OS)
 ELSE PRINT
11040 FOR K=1 TO OG(I,2):POKE OG(I,1)+K-1,PW(K)-256*(PW(K)<0):
NEXT
11050 FOR K=I TO OP-1:OL$(K)=OL$(K+1)
11060 FOR C=0 TO 2:OG(K,C)=OG(K+1,C):NEXT C,K
11070 OP=OP-1:I=I-1:GOTO 10960
11080 REM 1-byter zonder operand ----------
11090 IF A$(J+1)<>"" THEN ERROR 62
11100 PW(1)=W2((PO-1)/4):RETURN
11110 REM 2-byter zonder operand ----------
11120 IF A$(J+1)<>"" THEN ERROR 62
11130 PW(2)=W3((PO-1)/4):RETURN
11140 REM pseudo-instructies ----------
11150 J=J+1:OD$=A$(J):OU$=OD$
11160 ON (PO-1)/4 GOTO 11210,11250,11270,11290,11310,11360
11170 REM EQU ----------
11180 IF LA$="" THEN ERROR 63
11190 A$=OD$:GOSUB 13350
11200 WL(LT-1)=WA:RETURN
11210 REM ORG ----------
11220 IF OD$="" THEN ERROR 64
11230 A$=OD$:GOSUB 13350
11240 LP=0:MP=WA:MS=MP:RETURN
11250 REM END ----------
11260 GOTO 10590
11270 REM DB ----------
11280 A$=OD$:GOSUB 13610:RETURN
11290 REM DW ----------
11300 A$=OD$:GOSUB 13420:RETURN
11310 REM DM ----------
11320 IF LEFT$(OD$,1)<>CHR$(34) OR RIGHT$(OD$,1)<>CHR$(34)
 THEN ERROR 66
11330 TU$=MID$(OD$,2,LEN(OD$)-2)
11340 LP=LEN(TU$)
11350 FOR I=1 TO LP:PW(I)=ASC(MID$(TU$,I,1)):NEXT:RETURN
11360 REM ds ----------
11370 A$=OD$:GOSUB 13420
11380 DS=WA:LP=0:RETURN
11390 REM interpretatie van overige instructies ----------
11400 J=J+1:OD$=A$(J)
11410 OU$=OD$:TU=(PO+3)/4
11420 IF OD$="" AND IN$<>"RET " THEN ERROR 64
11430 GOSUB 13090
11440 PI=INSTR(OD$,"IX")
11450 IF PI<>0 THEN WI=&HDD:IR$="IX":GOTO 11520
```

```
11460 PI=INSTR(OD$,"IY")
11470 IF PI<>0 THEN WI=&HFD:IR$="IY":GOTO 11520
11480 ON TU GOTO 12620,11940,11920,12060,12060,12100,12220,
12240,12330,12310,12370,12420,12420,12510,12550
11490 IF TU<24 THEN 11630
11500 IF TU<32 THEN 11790
11510 GOTO 11850
11520 REM geindiceerde instructies ----------
11530 FI=-1
11540 IF (NOT HF) OR (PI-PA<>1) THEN MID$(OD$,PI,2)="HL":
GOTO 11600
11550 IF LEFT$(HI$,3)<>IR$+"+" THEN IF IN$="JP  " THEN 11590
 ELSE ERROR 60
11560 A$=RIGHT$(HI$,LEN(HI$)-3)
11570 DI$=A$:GOSUB 13610
11580 DF=-1:DW=WA
11590 OD$=LEFT$(OD$,PA)+"HL"+RIGHT$(OD$,LEN(OD$)-PZ+1)
11600 IF (INSTR(OD$,"IX")=0)AND(INSTR(OD$,"IY")=0) THEN 11620
11610 IF (OD$=("HL,"+IR$))AND(IN$="ADD ") THEN OD$="HL,HL"
 ELSE ERROR 61
11620 GOSUB 13090:GOTO 11480
11630 REM rekenkundige en logische instructies ----------
11640 IF NOT KF THEN A$=O1$:GOTO 11660
11650 IF O1$<>"A" THEN 11710 ELSE A$=O2$
11660 LP=1:CO=TU-16
11670 GOSUB 13260
11680 IF RF THEN PW(1)=128 OR (CO*8) OR RR:RETURN
11690 PW(1)=&HC6 OR(CO*8)
11700 GOSUB 13610:RETURN
11710 IF O1$<>"HL" THEN ERROR 61
11720 A$=O2$:GOSUB 13290
11730 IF NOT RF THEN ERROR 61
11740 IF IN$="ADD " THEN CO=9:LP=1:GOTO 11780
11750 PW(1)=&HED:LP=2
11760 IF IN$="ADC " THEN CO=&H4A:GOTO 11780
11770 IF IN$="SBC " THEN CO=&H42 ELSE ERROR 60
11780 PW(LP)=CO OR (DD*16):RETURN
11790 REM roteer- en schuifinstructies ----------
11800 LP=2:PW(1)=&HCB
11810 IF KF THEN ERROR 61
11820 A$=OD$:GOSUB 13260
11830 IF NOT RF THEN ERROR 61
11840 PW(2)=(8*(TU-24)) OR RR:RETURN
11850 REM bitmanipulatie-instructies ----------
11860 LP=2:PW(1)=&HCB
11870 A$=O2$:GOSUB 13260
11880 IF NOT RF THEN ERROR 61
11890 BB=ASC(O1$)-48
```

```
11900 IF (O>BB) OR (7<BB) OR (LEN(O1$)<>1) THEN ERROR 61
11910 PW(2)=(64*(TU-31))OR(BB*8) OR RR:RETURN
11920 REM relatieve sprongen ----------
11930 LP=1:PW(1)=&H10:A$=OD$:GOTO 12010
11940 LP=1
11950 IF NOT KF THEN CC=3:A$=OD$:GOTO 12000
11960 A$=O1$:GOSUB 13320
11970 IF (NOT CF) OR (CC>3) THEN ERROR 61
11980 CC=CC OR 4
11990 A$=O2$
12000 PW(1)=CC*8
12010 IF LEFT$(A$,1)<>"$" THEN GOSUB 13420:LP=LP-2:IF I>LT
 THEN WA=MP:GOTO 12030:ELSE 12030
12020 WA=MP+VAL(RIGHT$(A$,LEN(A$)-1))
12030 LP=LP+1:SO=MP:DA=WA
12040 GOSUB 13660
12050 PW(2)=OS:RETURN
12060 REM sprongen ----------
12070 TU=1:LP=1
12080 IF IN$="RET " THEN CO=0 ELSE CO=&B100
12090 GOTO 12130
12100 IF OD$="(HL)" THEN LP=1:PW(1)=&HE9:RETURN
12110 CO=2
12120 TU=0:LP=1
12130 IF IN$="RET " THEN IF OD$="" THEN 12150 ELSE 12170 ELSE
12140 IF KF THEN 12170
12150 PW(1)=&HC0 OR CO OR 1 OR (TU*8)
12160 A$=OD$:GOTO 12200
12170 A$=O1$:GOSUB 13320
12180 IF NOT CF THEN ERROR 61
12190 PW(1)=&HC0 OR CO OR (CC*8):A$=O2$
12200 IF IN$="RET " THEN RETURN
12210 GOSUB 13420:RETURN
12220 REM telinstructies ----------
12230 TU=0:GOTO 12250
12240 TU=1
12250 IF KF THEN ERROR 61
12260 LP=1:A$=OD$:GOSUB 13260
12270 IF RF THEN PW(1)=4 OR (RR*8) OR TU:RETURN
12280 GOSUB 13290
12290 IF NOT RF THEN ERROR 61
12300 PW(1)=3 OR (DD*16) OR (TU1*8):RETURN
12310 REM stackinstructies ----------
12320 CO=&HC1:GOTO 12340
12330 CO=&HC5
12340 A$=OD$:DR$(3)="AF":GOSUB 13290:DR$(3)="SP"
12350 IF NOT RF THEN ERROR 61
12360 LP=1:PW(1)=CO OR (DD*16):RETURN
```

```
12370 REM restart ----------
12380 A$=OD$:GOSUB 13610
12390 TU=WL/8
12400 IF TU<>INT(TU) OR TU>7 THEN ERROR 65
12410 LP=1:PW(1)=&HC7 OR (TU*8):RETURN
12420 REM I/O-instructies ----------
12430 IF NOT(KF AND HF) THEN ERROR 61
12440 IF IN$="IN  " THEN TU=0 ELSE TU=1:SWAP O2$,O1$
12450 IF HI$<>"C" THEN  12490
12460 A$=O1$:GOSUB 13260
12470 IF (NOT RF) OR (HI$<>"C") THEN ERROR 61
12480 LP=2:PW(1)=&HED:PW(2)=64 OR (RR*8) OR TU:RETURN
12490 LP=1:A$=HI$:GOSUB 13610
12500 PW(1)=&HDB XOR (TU*8):RETURN
12510 REM interruptmodi ----------
12520 IF OD$<>"0" AND OD$<>"1" AND OD$<>"2" THEN ERROR 65
12530 LP=2:PW(1)=&HED
12540 PW(2)=&H46 OR ((VAL(OD$)-(OD$<>"0"))*8):RETURN
12550 REM wisselinstructies ----------
12560 LP=1
12570 IF OD$="(SP),HL" THEN PW(1)=&HE3:RETURN
12580 IF OD$="DE,HL" THEN PW(1)=&HEB:GOTO 12610
12590 IF OD$="AF,AF'" THEN PW(1)=&H8:RETURN
12600 ERROR 61
12610 IF FI THEN ERROR 61 ELSE RETURN
12620 REM laadinstructies ----------
12630 IF NOT KF THEN ERROR 61
12640 A$=O1$:GOSUB 13260
12650 IF RF THEN 12850
12660 GOSUB 13290
12670 IF RF THEN 12750
12680 A$=O2$:GOSUB 13290
12690 IF RF THEN 12730
12700 SWAP O1$,O2$
12710 A=0:GOSUB 12930
12720 IF NF THEN ERROR 61 ELSE RETURN
12730 IF NOT HF THEN ERROR 61
12740 TU$=O2$:ZF=1:GOTO 12790
12750 IF OD$="SP,HL" THEN LP=1:PW(1)=&HF9:RETURN
12760 IF HF THEN TU$=O1$:ZF=0:GOTO 12790
12770 A$=O2$
12780 LP=1:CO=1:GOTO 12820
12790 A$=HI$
12800 IF TU$="HL" THEN LP=1:CO=&HA:GOTO 12820
12810 LP=2:PW(1)=&HED:CO=&H4B
12820 CO=CO AND NOT (ZF*8)
12830 PW(LP)=CO OR (DD*16)
12840 GOSUB 13420:RETURN
```

```
12850 ZZ=RR:A$=02$:GOSUB 13260
12860 IF NOT RF THEN 12890
12870 LP=1:PW(1)=64 OR(ZZ*8) OR RR
12880 IF PW(1)=&H76 THEN ERROR 61 ELSE RETURN
12890 A=1:GOSUB 12930
12900 IF NOT NF THEN RETURN
12910 LP=1:PW(1)=6 OR (RR*8)
12920 A$=02$:GOSUB 13610:RETURN
12930 REM 8-bits speciaal laden ----------
12940 NF=0
12950 IF 01$<>"A" THEN NF=-1:RETURN
12960 IF HF THEN 13020
12970 IF 02$="I" THEN TU=0:GOTO 13000
12980 IF 02$="R" THEN TU=1:GOTO 13000
12990 NF=-1:RETURN
13000 CO=&H47:LP=2:PW(1)=&HED
13010 PP=(A*2) OR TU:GOTO 13070
13020 IF HI$="BC" THEN TU=0:GOTO 13060
13030 IF HI$="DE" THEN TU=1:GOTO 13060
13040 LP=1:PW(1)=&H32 OR (A*8)
13050 A$=HI$:GOSUB 13420:RETURN
13060 CO=2:LP=1:PP=(TU*2) OR A
13070 PW(LP)=CO OR (8*PP):RETURN
13080 REM subroutines ----------
13090 REM operand ontleden ----------
13100 PO=INSTR(OD$,",")
13110 IF PO=0 THEN 01$=OD$:KF=0:GOTO 13140
13120 KF=-1
13130 01$=LEFT$(OD$,PO-1):02$=RIGHT$(OD$,LEN(OD$)-PO)
13140 PA=INSTR(OD$,"("):PZ=INSTR(OD$,")")
13150 IF PA=0 THEN HF=0:HI$="":GOTO 13190
13160 IF PA>PZ THEN ERROR 61
13170 HF=-1 .
13180 HI$=MID$(OD$,PA+1,PZ-PA-1)
13190 RETURN
13200 REM labeltest ----------
13210 LC=ASC(A$)
13220 IF LC<65 OR LC>90 THEN NO=-1:RETURN
13230 IF LEN(A$)>6 THEN PRINT"Label te lang":A$=LEFT$(A$,6)
13240 LA$=A$:NO=0
13250 RETURN
13260 REM registertest ----------
13270 FOR I=0 TO 7:IF RG$(I)=A$ THEN RF=-1:RR=I:RETURN ELSE NEXT
13280 RF=0:RETURN
13290 REM test registerpaar ----------
13300 FOR I=0 TO 3:IF DR$(I)=A$ THEN RF=-1:DD=I:RETURN ELSE NEXT
13310 RF=0:RETURN
13320 REM voorwaardetest ----------
```

```
13330 FOR I=0 TO 7:IF CD$(I)=A$ THEN CF=-1:CC=I:RETURN ELSE NEXT
13340 CF=0:RETURN
13350 REM getaltest ----------
13360 WA=VAL(A$)
13370 LC=ASC(A$)
13380 IF WA=0 AND LC<>38 AND (LC>57 OR LC<48) THEN ERROR 66
13390 IF WA>=0 THEN RETURN
13400 IF LEFT$(A$,2)<>"&H" THEN ERROR 65
13410 RETURN
13420 REM term interpreteren ----------
13430 GOSUB 13200
13440 IF NO THEN GOSUB 13350:GOTO 13480
13450 FOR I=0 TO LT:IF LT$(I)<>LA$ THEN NEXT
13460 IF I>LT THEN 13540
13470 WA=WL(I)
13480 WH=INT(WA/256)
13490 WL=WA-WH*256
13500 LP=LP+2
13510 PW(LP-1)=WL
13520 PW(LP)=WH
13530 RETURN
13540 IF LF THEN PRINT"?";
13550 OL$(OP)=A$:OG(OP,0)=RN
13560 OG(OP,1)=MP+LP-FI-DF
13570 OG(OP,2)=2+(IN$="DJNZ" OR IN$="JR  ")
13580 OP=OP+1
13590 WA=0
13600 GOTO 13480
13610 REM term <256 interpreteren ----------
13620 GOSUB 13420
13630 LP=LP-1
13640 IF WH<>0 THEN PRINT WH:ERROR 65
13650 RETURN
13660 REM offset berekenen ----------
13670 OS=DA-SO
13680 OS=OS-2
13690 IF OS>129 OR OS<-126 THEN ERROR 65
13700 IF OS<0 THEN OS=256+OS
13710 RETURN
13720 REM initialiseren ----------
13730 DEFINT A-Z:R$=";":PO=0:J=0
13740 VP=VARPTR(R$)
13750 DEF FNDK(I)=VAL("&H"+HEX$(PEEK(I)+256*PEEK(I+1)))
13760 SCREEN 0:WIDTH 39:COLOR 1,14,14:KEY OFF
13770 T1$="LD  JR  DJNZCALLRET JP  INC DEC PUSHPOP RST IN  OUT
 IM  EX  ADD ADC SUB SBC AND XOR OR  CP  RLC RRC RL  RR  SLA SRA
 *** SRL BIT RES SET "
13780 T3$="CPD CPDRCPI CPIRIND INDRINI INIRLDD LDDRLDI LDIRNEG
```

```
 OTDROTIROUTDOUTIRETIRETNRLD RRD "
13790 DATA A9,B9,A1,B1,AA,BA,A2,B2,A8,B8,A0,B0,44,BB,B3,AB,A3,
4D,45,6F,67
13800 T2$="CCF CPL DAA DI  EI  EXX HALTNOP RLA RLCARRA RRCASCF "
13810 DATA 3F,2F,27,F3,FB,D9,76,00,17,07,1F,0F,37
13820 T4$="EQU ORG END DB  DW  DM  DS  "
13830 DIM A$(3),LT$(50),WL(50),OL$(50),OG(50,2)
13840 DIM W2(12),W3(20)
13850 FOR I=0 TO 20:READ A$:W3(I)=VAL("&H"+A$):NEXT
13860 FOR I=0 TO 12:READ A$:W2(I)=VAL("&H"+A$):NEXT
13870 BP=FNDK(FNDK(&HF676)):MP=&HF000:MS=MP
13880 DIM RG$(7),CD$(7),DR$(3)
13890 FOR I= 0 TO 7:READ RG$(I):NEXT
13900 FOR I= 0 TO 7:READ CD$(I):NEXT
13910 FOR I= 0 TO 3:READ DR$(I):NEXT
13920 DATA B,C,D,E,H,L,(HL),A
13930 DATA NZ,Z,NC,C,PO,PE,P,M
13940 DATA BC,DE,HL,SP
13950 ON STOP GOSUB 10780:STOP ON
13960 ON ERROR GOTO 14080
13970 FOR I=&HF374 TO &HF37F:READ A$:POKE I,VAL("&H"+A$):NEXT
13980 DATA F5,3A,61,F6,32,15,F4,F1
13990 DATA CD,63,1B,C9
14000 POKE &HFEE5,&H74:POKE &HFEE6,&HF3
14010 FOR I=&HF100 TO &HF122:READ A$:POKE I,VAL("&H"+A$):
NEXT:DEFUSR9=&HF100
14020 DATA FE,03,C2,6D,40,1A,FE,00
14030 DATA C8,47,62,6B,23,7E,23,66
14040 DATA 6F,7E,FE,7B,30,06,FE,61
14050 DATA 38,02,E6,DF,77,23,10,F1
14060 DATA 3E,03,C9
14070 RETURN
14080 REM foutenbehandeling ----------
14090 IF ERR<60 THEN POKE &HFEE4,&HC9:KEY ON:ON ERROR GOTO 0
14100 BEEP:FA=FA+1
14110 IF NOT LF THEN LOCATE 5,3:PRINT RN:PRINT
14120 IF ERR=60 THEN PRINT"syntax error ";
14130 IF ERR=61 THEN PRINT"syntax error in operand ";
14140 IF ERR=62 THEN PRINT"operand teveel ";
14150 IF ERR=63 THEN PRINT"label mist ";
14160 IF ERR=64 THEN PRINT"operand mist ";
14170 IF ERR=65 THEN PRINT"waarde niet toegelaten ";
14180 IF ERR=66 THEN PRINT"teken niet toegestaan ";
14190 PRINT" van ";ERL;
14200 PRINTTAB(30);RN,RA$
14210 RESUME 10490
```

4.9.2 Programmabeschrijving

Namen van variabelen staan in dit gedeelte tussen aanhalingstekens.

regel 1

De adressen &HF000-&HF31F in het RAM worden gereserveerd voor het machinetaalprogramma. Vervolgens springt het programma over het source-programma heen (regel 2-9999).

regel 10010

Sprong naar programma-onderdeel initialisering: onder andere de instructietabel wordt gevormd (zie regel 13720 e.v.).

regel 10020-10090

Menu: listflag "LF" en uitvoerapparaat (beeldscherm of printer) worden gedefinieerd.

regel 10100-10160

De BP (BASIC program pointer) wijst op het actuele adres in het BASIC source-programma. Aan het begin van een regel staat de lengte van de BP als low- en highbyte. FNDK(BP) leest de 16-bits waarde op adres BP en BP+1. De waarde komt overeen met het beginadres van de volgende regel, "NR". De BP wordt verhoogd met 2 en het regelnummer "RN" wordt gelezen. Is dit groter dan 9999, dan beëindigt het programma de vertaling. In regel 10150 test het programma of de apostrof aan het begin van de regel staat. Is dat niet het geval, dan volgt een foutmelding en wordt de volgende regel gelezen.

regel 10170-10190

Dit programma-onderdeel vult "R$" met de actuele regel. Om de snelheid van de assembler zo hoog mogelijk te houden gebeurt dit via een verandering van de stringwijzer van "R$" in de interne variabelentabel.

regel 10200-10360

Het programma slaat eerst eventueel commentaar op in "OP$". Vervolgens wordt de overgebleven regel bij elke spatie gesplitst en worden de gesplitste delen in "A$(J)" opgeslagen. Wanneer de regel in meer dan drie delen wordt ontleed (label, instructie, operand), dat wil zeggen J>2, is er sprake van een syntax error.

regel 10370

Sprong naar de subroutine die de interpretatie uitvoert.

regel 10380-10580

Uitvoer: is "FI" (flag voor geïndiceerde instructies) gezet dan springt het programma eerst naar de extra routine die begint in regel 10520. In andere gevallen voert het programma de complete assemblerregel uit. Voordat het programma naar het begin springt om de volgende regel te vertalen, reset het de relevante variabelen en verhoogt het de MP (machine program counter) met "LP", de lengte van de instructie.

regel 10590-10780

Dit deel voert aan het eind van het programma de ongedefinieerde labels uit: programmanaam, begin- en eindadres, lengte, aantal fouten en variabelentabel. Vanaf regel 10770 slaat het de objectcodes op.

regel 10790-10860

Deze regels testen of "A$(J)" een geldige instructie is. Zo ja, dan springt het programma naar de plaats waar dergelijke instructies worden vertaald.

regel 10870-10880

Is er geen instructie gevonden, dan testen deze regels of er pseudo-instructies zijn en springen daar naartoe.

regel 10890-11070

Gaat het om een label, dan wordt dit in de labeltabel gezet en krijgt het label de waarde MP (machine program counter) (regel 10910-10940). De volgende regels tot en met 11070 testen of het label al eens is gebruikt op een moment dat het nog niet was gedefinieerd. Klopt dat, dan wordt alsnog de betreffende waarde gePOKEt en het label uit de tabel van de ongedefinieerde labels "OL$(I)" gewist. Gaat het om een ongeldig label (zonder letter aan het begin), dan verschijnt de foutmelding "syntax error" (regel 10920).

regel 11080-11100

Deze regels interpreteren instructies met een opcode van 1 byte die geen operand bezitten. De code volgt uit de plaats in "T2$" en de bijbehorende "W2(I)".

regel 11110-11130

Deze regels behandelen de 2-bytes instructies zonder operand. De eerste opcode is altijd &HED (PW(1)=&HED). De tweede byte van de opcode volgt uit de plaats van de instructie in "T3$" en "W3$(I)".

regel 11140-11380

Hier worden alle pseudo-instructies vertaald.

regel 11390-13070

Wanneer de instructie in geen van de genoemde groepen voorkomt, wordt hij hier geïnterpreteerd.

regel 11430

Hier wordt de operand gesplitst.

regel 11440-11470

Gaat het hier om een geïndiceerd geadresseerde instructie dan vertakt het programma naar regel 11520.

regel 11480-11510

Op grond van de plaats in "TI$" springt het programma naar de betreffende routine die de instructie behandelt.

regel 11520-11620

HL vervangt bij geïndiceerde instructies XY of XY+dis (XY is IX of IY). "FI" en "DI" worden dienovereenkomstig ingesteld. Het programma loopt normaal verder vanaf regel 11480. Na de interpretatie wordt de verandering teniet gedaan en de code van de geïndiceerde instructie (die analoog is aan die van HL) uitgevoerd.

regel 11630-11780

Deze regels interpreteren de 8- en 16-bits rekeninstructies.

regel 11790-11840

Roteer- en schuifinstructies.

regel 11850-11910

Bitmanipulatie-instructies.

regel 11920-12050

Relatieve sprongen (JR en DJNZ).

regel 12060-12210

Andere sprongen (JP,RET,CALL).

regel 12220-12300

Telinstructies (INC,DEC).

regel 12310-12620

Zie REM-regels.

regel 12630-13070

Laadinstructies

Het zou te ver voeren om hier alle routines uit te leggen, daarom nemen we als voorbeeld de routine voor de bitmanipulatie-instructies.

regel 11860

"LP" (lengte van de instructie: aantal waarden dat moet worden gePOKEt) is 2. De eerste waarde, "PW(1)", is &HCB. Dit geldt voor alle bitinstructies.

regel 11870

Het gedeelte van de operand na de komma ("O2$") wordt in "A$" opgeslagen om aan de subroutine te worden overgedragen die begint in regel 13260. De subroutine controleert of het om een van de registers A, B, C, D, E, H, L of (HL) gaat.

regel 11880

Is er geen overeenstemming gevonden (RF=0) dan wordt de foutmelding "syntax error in operand" uitgevoerd. In de andere gevallen wordt de code van het register teruggegeven aan RR en is RF gezet.

regel 11890

BB krijgt de waarde van het getal voor de komma (van het bitnummer).

regel 11900

Deze regel test of het getal tussen 0 en 7 ligt. Is dit niet het geval dan verschijnt weer "syntax error in operand".

regel 11910

Tenslotte wordt de opcode opgeslagen in "PW(2)". De opcode is als

volgt samengesteld:

01 BB RR voor BIT-instructies
10 BB RR voor RES-instructies
11 BB RR voor SET-instructies

Uit ("TU"-31)*64 resulteren de bits 7 en 6 van de opcode. "TU" is de positie van de instructie in "T1$". BB*8 staat voor de bits 5-3 en RR voor de bits 2-0. De subroutine heeft vastgesteld wat RR is; de waarde komt overeen met de registercode. Is de opcode berekend dan wordt naar de uitvoer (regel 10380) gesprongen. De andere routines functioneren net zo.

regel 13080-14150

Hier staan vaak gebruikte subroutines:

- 13090-13190	Eerst kijkt het programma of de operand "OP$" een komma bevat. Is dit het geval dan wordt de operand gesplitst in een gedeelte voor de komma, "O1$", en een gedeelte na de komma, "O2$". "KF" wordt op -1 (waar) gezet. Daarna test het programma of er haakjes zijn. Zo ja dan wordt de inhoud tussen haakjes in "H1$" opgeslagen en wordt "HF" gezet (-1).
- 13200-13250	Test of "A$" een toelaatbaar label bevat.
- 13260-13280	Test of "A$" een register bevat (A, B, C, D, E, H, L, (HL)).
- 13290-13310	Test of "A$" een van de registerparen BC, DE, HL of SP bevat.
- 13320-13340	Test of "A$" een van de voorwaarden C, NC, Z, NZ, PO, PE, P of M bevat.
- 13350-13410	Geeft de waarde van "A$", als dit een getal is.
- 13420-13600	Zoekt de 2-bytes waarde van "A$"; "A$" kan een getal zijn, een variabele of een label.
- 13610-13650	Zoekt de 1-bytes waarde (lowbyte) van "A$".
- 13660-13710	Berekent de offset voor relatieve sprongen.

regel 13720-14390

Initialisering: de velden met gegevens en strings voor vergelijking worden geproduceerd. "VP" wijst op het adres waar de lengte van "R$" staat. FNDK(X) geeft de 16-bits waarde aan van twee opeenvolgende geheugenplaatsen. BP wordt aan het begin van de regel gezet die volgt op regel 1. MP wordt op &HF000 gezet.

regel 13970-14000

Hier vindt de verandering in het bedrijfssysteem plaats die nodig is voor uitvoer via printer en beeldscherm.

regel 14010-14060

Definitie van de functie USR9: de kleine letters in de string worden veranderd in hoofdletters.

regel 14080-14210

Behandeling van fouten: bij een "gewone" fout (ERR<60) wordt de printeruitvoer uitgeschakeld en de bijbehorende foutmelding uitgevoerd.

Variabelenlijst

SUB betekent subroutine

A	overdragen aan SUB 8-bits speciaal laden
A$	overdragen aan diverse subroutines
BB	bitnummercode bij bitmanipulatie-instructies
BP	BASIC programmawijzer (BASIC program pointer)
CC	conditiecode bij sprongen
CF	gezet (-1) indien de voorwaarde is gevonden en gereset (0) indien niet gevonden. SUB cond test levert de voorwaarde
CO	gebruikt om opcodes te maken van de betreffende instructie
DA	doeladres
DI$	bevat de afstand (distance) bij geïndiceerde instructies
DI	gezet bij een geïndiceerde instructie met afstandsaanduiding, in andere gevallen gereset
DS	bevat het aantal door een DS-instructie gereserveerde geheugenplaatsen
DW	waarde van de afstand (twee-complement)
FA	aantal fouten
FI	gezet bij geïndiceerde instructies, anders gereset

HF	gezet indien haakjes in de operand, anders gereset
HI$	bevat de inhoud van de haakjes van de operand (mits voorhanden)
I,J,C	lustellers
IN$	assemblerinstructie
IR$	bevat IX of IY bij geïndiceerde adressering
KF	gezet indien komma in operand, anders gereset
LA$	SUB labeltest levert labelnaam
LB$	actuele labelnaam
LC	ASCII-code van eerste teken van te testen label (SUB labeltest)
LF	gezet indien een listing is gewenst, anders gereset
LP	lengte van de betreffende instructie (lengte van de objectcode)
LT	wijzer op de vrije plaats in de labeltabel (LT$)
MP	machine program counter: wijst op adres waar volgende machinetaalcode komt te staan
MS	beginadres van machinetaalprogramma
NA$	programmanaam
NF	gezet indien er bij een laadinstructie sprake is van onmiddellijke adressering, anders gereset (SUB 8-bits speciaal laden)
NO	no label flag: gezet bij labeltest zonder resultaat, anders gereset; SUB labeltest levert dit gegeven
O1$	gedeelte van de operand voor de komma
O2$	gedeelte van de operand achter de komma
OF	waarde die SUB offset heeft berekend
OD$	operand die moet worden verwerkt
OP	ongedefinieerde label tabel pointer: wijst op de volgende vrije plaats in de tabel OT$ of OG$
OP$	opmerking (commentaar) bij een assemblerregel
OU$	operand, in eerste instantie bedoeld voor uitvoer
PA	positie van (in operand
PO	positie van instructie in de test-strings
PZ	positie van) in operand
R$	bevat de actuele regel in behandeling
RA$	in eerste instantie gereserveerd voor actuele regel
RF	gezet als SUB een register (A,B,C,D,E,H,L,(HL)) heeft waargenomen, anders gereset
RN	actuele regelnummer
RR	code van het register; SUB levert deze waarde
SO	overdragen aan SUB offset berekenen: beginadres
TU1	diverse bufferregistraties
TU$	diverse bufferregistraties

T1$	test overige: bevat alle instructies met operand
T2$	test alle instructies met een opcode van 1 byte zonder operand
T3$	test &HED; bevat alle instructies met een opcode van 2 bytes zonder operand; de eerste byte is &HED
T4$	test pseudo; bevat alle pseudo-instructies
VP	variable pointer; wijst op het adres van R$ in de interne variabelentabel
WA	waarde van een term, geleverd door SUB waarden of SUB getaltest
WH	highbyte van de waarde
WL	lowbyte van de waarde

Tabellen

LT$(50)	labeltabel
WL(50)	waarden van de labels uit de waardentabel
OL$(50)	lijst van ongedefinieerde labels
OG(50,2)	gegevens bij de ongedefinieerde labels
OG(I,0)	nummer van regel waarin ongedefinieerde labels voorkomen
OG(I,1)	adres van de waarde die nog met POKE moet worden ingevoegd
OG(I,2)	type, dat wil zeggen 16-bits (2) of offset (1)
W2(12)	opcodes van de 1-bytes instructies (T2$)
W3(20)	opcodes van de 2-bytes instructies (T3$)
RG$(7)	registertabel: B, C, D, E, H, L, (HL), A
CD$(7)	conditietabel: NZ, Z, NC, C, PO, PE, A, M
DR$(3)	tabel met registerparen: BC, DE, HL, SP

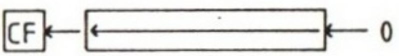
4.10 Roteer- en schuifinstructies

Cijfers verschuiven in een getal - wat houdt dat in?

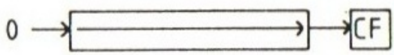
10^4	10^3	10^2	10^1	10^0	
	3	7	3	0	
3	7	3	0	0	naar links
		3	7	3	naar rechts

In het decimale stelsel betekent een verschuiving naar links (komma naar rechts) een vermenigvuldiging met 10 (grondtal) en een verschuiving naar rechts (komma naar links) een deling door 10. Zo betekent een verschuiving in het binaire stelsel delen door of vermenigvuldigen met twee. In BASIC zijn er voor deze instructies geen directe equivalenten (behalve dan vermenigvuldigen met of delen door 2). De Z80 kent 76 van dergelijke instructies, waarvan de meeste impliciete, indirecte of geïndiceerde adressering gebruiken. Er zijn verschillende manieren om te roteren of te verschuiven. Eerst maar eens iets over het onderscheid tussen verschuiving en roteren.

Verschuiven: bij het verschuiven naar rechts of links beweegt de inhoud van het register bit voor bit. De bit die aan de ene kant wegvalt, komt in de carry. Op de vrije plaats aan de andere kant van de byte komt een 0.



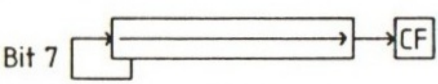
SLA: rekenkundige verschuiving naar links



SRL: logische verschuiving naar rechts

afbeelding 7

De instructie hiervoor is SRL. Bij getallen met een voorteken levert deze instructie een fout op. Bit 7, de sign-bit, wordt naar de plaats van bit 6 verschoven. Op de plaats van bit 7 komt een 0. Zo zou een negatief getal (bit 7=1) een positief getal (bit 7=0) worden. Met SRA voorkomen we deze fout. Bij deze instructie is de links toegevoegde bit gelijk aan de voortekenbit. Deze is 0 als de linker bit 0 (+) is en 1 als de linker bit 1 (-) is. We noemen SRA een rekenkundige (en niet een logische) schuifinstructie, omdat de rekenkundige betekenis van de 7e bit niet verandert.



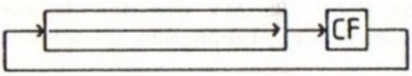
SRA: rekenkundige verschuiving naar rechts

afbeelding 8

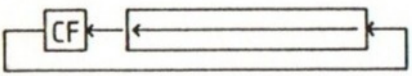
Roteren: anders dan bij verschuivingen is bij rotaties de toegevoegde bit gelijk aan de bit die uit de boot valt of de carry-bit. De Z80 kent twee soorten rotaties:

8-bits rotatie (zonder carry)
9-bits rotatie (met carry)

Bij een 9-bits rotatie naar rechts verschuiven alle 8 bits 1 plaats naar rechts. De bit die rechts uit de boot valt, komt in de carry. De bit die aan de linkerkant binnenkomt is de oude inhoud van de carry. De 8 bits van de byte plus de carry worden geroteerd, vandaar de term 9-bits rotatie.



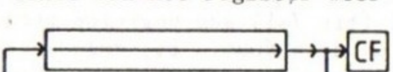
RRC: roteer rechts via carry



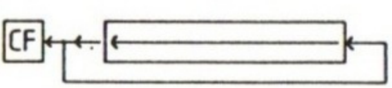
RLC: roteer links via carry

afbeelding 9 9-bits rotatie

Bij de 8-bits rotatie roteren alleen de 8 bits van het register. In de carry wordt de weggeschoven bit opgeslagen. De oude inhoud van de carry doet niet mee. De weggeschoven bit komt aan het andere einde van het register weer terug.



RR: roteer rechts



RL: roteer links

afbeelding 10 8-bits rotatie

Er zijn twee speciale instructies om cijfers (blokken van 4 bits) in BCD-formaat te roteren. RLD en RRD (D staat voor digit (cijfer)) roteren twee cijfers van de geheugenplaats waar HL op wijst, plus het cijfer dat correspondeert met de onderste helft van de accu. Roteer- en schuifinstructies hebben meestal een opcode van 2 bytes. De eerste byte van de opcode is altijd &HCB. Bij geïndiceerd geadresseerde instructies is &HCB de tweede byte, omdat de eerste byte bij deze wijze van adresseren &HDD of &HFD is (uitzondering: alleen RRD/RLD beginnen met &HED). Omdat u roteerinstructies vaak nodig heeft voor rekenroutines, zijn er nog vier andere instructies. Deze hebben alleen betrekking op de accu en hebben een opcode van 1 byte. Ze zijn de helft korter dan standaardinstructies en twee keer zo snel.

normaal	speciaal voor accu
RLC A	RLCA
RRC A	RRCA
RL A	RLA
RR A	RRA

De normale roteer- en schuifinstructies beïnvloeden de S- en Z-flag op de normale wijze. De P/V-flag geeft de pariteit aan. De inhoud van de carry is steeds de weggeschoven bit. De speciale instructies voor de accu veranderen S, Z en P/V niet. De BCD-roteerinstructies RLD/RDD beïnvloeden alleen de S-, Z- en P-flag zoals boven beschreven, en niet de carry. Een paar voorbeelden:

SRL C

```
C=&H36=&B00110110

0 --> 0011011 0 --> naar carry
      00011011

&B00011011=&H1B  C-register na afloop
0                CF (carry flag) na afloop
```

SRL deelt door 2: &H36/2=&H1B

SRA (HL)

```
HL=&HB100
&HB100 bevat &HC2=&B11000010

 1100001 0 --> naar carry
11100001
```

De 0 verdwijnt naar de carry en de oude voortekenbit (bit 7) blijft gehandhaafd:

```
&B11100001=&HE1  (HL) na afloop
0                CF na afloop
```

Als twee-complement is

```
&HC2 = -62
&HE1 = -31
```

SRA halveert getallen met een voorteken correct. SRL (HL) zou als resultaat &H61 (97) hebben. Dit is niet de helft van -62, maar de helft van 194, wat overeenkomt met &HC2 als getal zonder voorteken.

RLC D

```
D=&HE4=&B11100100
CF=1

nieuwe carry  <--  1 1100100  <--  1  vorige carry
                     11001001

&B11001001=&HC9  D-register na afloop
1                CF na afloop
```

&HC9 is natuurlijk niet het dubbele van &HE4. De reden is dat een bit naar de carry is geroteerd. Dus moet &H1C9 het dubbele van &HE4 zijn. Dit is niet juist, omdat de oude carry (1) is geroteerd. Dus is &H1C9-1=&H1C8 het dubbele van &HE4. Roteert u getallen die uit meerdere bytes bestaan, dan roteert RLC of RRC de bit die bij de eerder geroteerde byte is weggeschoven via de carry in de volgende byte. Het programma aan het einde van deze paragraaf illustreert dit.

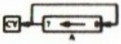
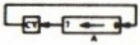
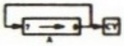
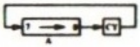
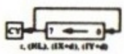
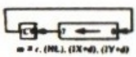
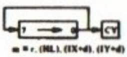
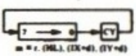
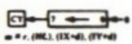
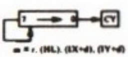
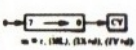
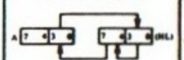
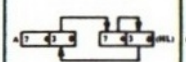
RRA

```
accu=&H76=&B01110110
CF=0

vorige carry 0  -->  0111011 0  -->  nieuwe carry
                     00111011

&B00111011=&H3B  accu-inhoud na afloop
0                CF na afloop

&H3B*2=&H76
```

Mnemonic	Symbolic Operation	C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
RLCA		↕	•	•	•	0	0	00 000 111	1	1	4	Rotate left circular accumulator
RLA		↕	•	•	•	0	0	00 010 111	1	1	4	Rotate left accumulator
RRCA		↕	•	•	•	0	0	00 001 111	1	1	4	Rotate right circular accumulator
RRA		↕	•	•	•	0	0	00 011 111	1	1	4	Rotate right accumulator
RLC r		↕	↕	P	↕	0	0	11 001 011 00 000 r	2	2	8	Rotate left circular register r
RLC (HL)		↕	↕	P	↕	0	0	11 001 011 00 000 110	2	4	15	r Reg.
RLC (IX+d)		↕	↕	P	↕	0	0	11 011 101 11 001 011 ← d → 00 000 110	4	6	23	000 B 001 C 010 D 011 E 100 H 101 L 111 A
RLC (IY+d)		↕	↕	P	↕	0	0	11 111 101 11 001 011 ← d → 00 000 110	4	6	23	
RL m		↕	↕	P	↕	0	0	010				Instruction format and states are as shown for RLC,m. To form new OP-code replace 000 of RLC,m with shown code
RRC m		↕	↕	P	↕	0	0	001				
RR m		↕	↕	P	↕	0	0	011				
SLA m		↕	↕	P	↕	0	0	100				
SRA m		↕	↕	P	↕	0	0	101				
SRL m		↕	↕	P	↕	0	0	111				
RLD		•	↕	P	↕	0	0	11 101 101 01 101 111	2	5	18	Rotate digit left and right between the accumulator and location (HL).
RRD		•	↕	P	↕	0	0	11 101 101 01 100 111	2	5	18	The content of the upper half of the accumulator is unaffected

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

Toepassing

De standaardtoepassing van rotatie- en schuifinstructies is binair vermenigvuldigen en delen. We behandelen nu een programma voor de vermenigvuldiging van twee 8-bits getallen. Eerst een vermenigvuldiging in het decimale stelsel:

```
 101
  29*
 ---
 909
2020
----
2929
```

In het tweetallig stelsel gaat het precies hetzelfde.

```
101 = &H65 = &B01100101
 29 = &H1D = &B00011101

      01100101
      00011101*
      --------
      01100101
             0
    01100101
   01100101
  01100101
 01100101
     0
    0
   0
-------------
0101101110001 = &HB71 = 2929
```

Het makkelijke is dat we in het tweetallig stelsel alleen vermenigvuldigen met 0 (resultaat = 0) en 1 (resultaat = factor). Daardoor kan de vermenigvuldiging van willekeurige getallen worden teruggebracht tot schuiven en optellen. Het belang van de schuifprocedure blijkt duidelijk uit de hierboven toegepaste werkwijze. Zo komen we tot het volgende programma.

```
10 ' ORG &HF000
20 ' LD B,B
30 ' LD A,(FAC1)
40 ' LD E,A
```

```
50 ' LD A,(FAC2)
60 ' LD D,0
70 ' LD HL,0
80 ' MULTIP RRCA
90 ' JR NC,NOADD
100 ' ADD HL,DE
110 ' NOADD SLA E
120 ' RL D
130 ' DJNZ MULTIP
140 ' LD (RESUL),HL
150 ' RET
160 ' FAC1 DB 101
170 ' FAC2 DB 29
180 ' RESUL DS 2
190 ' END
```

In dit source-programma voor vermenigvuldiging komen aan het eind de twee nieuwe pseudo-instructies DB en DS voor.

DB Define Byte

slaat de achter DB vermelde byte op op het actuele adres.

DS Define Storage

reserveert het achter DS vermelde aantal geheugenplaatsen. De adresteller wordt met dat aantal verhoogd. In dit geval gebruiken we de gereserveerde ruimte om het resultaat op te slaan. Voer het programma in en laat het vertalen door de assembler. Om de werking van het programma goed door te krijgen is het misschien verhelderend het programma eens 'op eigen kracht' na te doen. Neem de getallen uit het voorbeeld en voer iedere instructie uit op papier. Noteer na iedere stap het resultaat, dus de inhoud van de registers en de flags. Het eindresultaat moet natuurlijk 2929 (= &H1B7) zijn.

4.11 Goochelen met bits

In paragraaf 4.8 gingen we in op de mogelijkheid logische operaties te gebruiken om bits in de accu afzonderlijk of groepsgewijs te zetten of te resetten. Hiervoor bestaan bitmanipulatie-instructies, waarmee u een willekeurige bit in een

willekeurig register of geheugenplaats kunt zetten of resetten. De meeste processoren kennen maar weinig van deze instructies omdat ze gecompliceerd zijn. Bij de Z80 ligt dat anders. Met inbegrip van de instructies om bits te testen staan er 120 instructies ter beschikking waarmee bits kunnen worden gemanipuleerd. De testinstructies controleren of een bepaalde bit in een register of geheugenplaats is gezet of gereset. Afhankelijk van het resultaat wordt de zero flag gezet of gereset. De carry wordt niet beïnvloed, S-flag en P/V-flag zijn na afloop onbepaald. De instructies om de bit te zetten (SET) en te resetten (RES) hebben geen invloed op de flags. Alle bitinstructies beginnen met de opcode &HCB (behalve wanneer ze geïndiceerd geadresseerd zijn). De tweede opcode volgt uit het bitnummer en de registercode. De betreffende bytes kunnen op drie manieren worden geadresseerd:

- impliciet register A, B, C, D, E, H, L
- indirect (HL)
- geïndiceerd (XY+d)

formaat:

BIT b,r	BIT b,(HL)	BIT b,(XY+d)
SET b,r	SET b,(HL)	SET b,(XY+d)
RES b,r	RES b,(HL)	RES b,(XY+d)

(b=bitnummer)

Het bitnummer (b) is als volgt gecodeerd:

0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

Dergelijke instructies noemen we bit-geadresseerd, omdat de bit waar het om gaat in de opcode wordt aangegeven. Een paar voorbeelden:

BIT 6,B

B=&H33=&B00110011

Omdat bit 6 = 0, wordt de Z-flag gezet.

B is na afloop &H33.

flags:

S X
Z 1
V X
C

RES 1,(HL)

HL=&HA975
&HA975 bevat &H23=&B00100011

Bit 1 wordt gereset:

&B00100001=&H21

Geheugenplaats &HA975 bevat na afloop &H21.

flags:

S
Z
V
C

SET 7,C

C=&H7F=&B01111111

Bit 7 wordt gezet:

&B11111111=&HFF

C-flag is na afloop &HFF

flags:

S
Z
V
C

BASIC-equivalenten

Probeer SET b,A in BASIC toe te passen: bit b moet worden gezet. Met OR kunt u bepaalde bits zetten. De waarde van bit b is 2^b. Hieruit volgt:

assembler	BASIC
SET b,A	A=A OR (2^b)

Voor RES geldt hetzelfde:

RES b,A	A=A AND (255-2^b)

SCF en **CCF**

Er zijn twee speciale instructies omdat bit 0 in het F-register (de carry) vaak wordt gebruikt. SCF zet de carry flag. CCF levert het complement van de waarde van de carry flag. Dit zijn de enige instructies die de flags direct beïnvloeden. Met de logische instructies kunt u de carry wissen. SCF en CCF staan in de lijst aan het eind van de volgende paragraaf.

4.12 Stuurinstructies

Stuurinstructies veranderen of beïnvloeden de modus of het proces in de CPU.

NOP

NOP betekent no operation. Het is dus een instructie zonder functie. Dat klinkt vreemd maar heeft toch zijn nut. U kunt NOP namelijk gebruiken als opzettelijke vertrager (NOP duurt bij de MSX 1 microseconde (10^-6sec)) of als opvulling in een programma. Fouten opsporen en verwijderen wordt daardoor een stuk makkelijker. De opcode van NOP is &H00. Als het programma per ongeluk in een gewist bereik komt, wordt dit bereik niet veranderd of vernietigd omdat NOP geen veranderingen teweegbrengt.

Mnemonic	Symbolic Operation	C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
BIT b, r	$Z \leftarrow \overline{r}_b$	•	↕	X	X	0	1	11 001 011 01 b r	2	2	8	
BIT b, (HL)	$Z \leftarrow \overline{(HL)}_b$	•	↕	X	X	0	1	11 001 011 01 b 110	2	3	12	
BIT b, (IX+d)	$Z \leftarrow \overline{(IX+d)}_b$	•	↕	X	X	0	1	11 011 101 11 001 011 ← d → 01 b 110	4	5	20	
BIT b, (IY+d)	$Z \leftarrow \overline{(IY+d)}_b$	•	↕	X	X	0	1	11 111 101 11 001 011 ← d → 01 b 110	4	5	20	
SET b, r	$r_b \leftarrow 1$	•	•	•	•	•	•	11 001 011 [11] b r	2	2	8	
SET b, (HL)	$(HL)_b \leftarrow 1$	•	•	•	•	•	•	11 001 011 [11] b 110	2	4	15	
SET b, (IX+d)	$(IX+d)_b \leftarrow 1$	•	•	•	•	•	•	11 011 101 11 001 011 ← d → [11] b 110	4	6	23	
SET b, (IY+d)	$(IY+d)_b \leftarrow 1$	•	•	•	•	•	•	11 111 101 11 001 011 ← d → [11] b 110	4	6	23	
RES b, m	$s_b \leftarrow 0$ $m \equiv r$, (HL), (IX+d), (IY+d)							[10]				To form new OP-code replace [11] of SET b,m with [10]. Flags and time states for SET instruction

r	Reg.
000	B
001	C
010	D
011	E
100	H
101	L
111	A

b	Bit Tested
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Notes: The notation s_b indicates bit b (0 to 7) or location s.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

HALT

Deze instructie onderbreekt de operaties van de CPU net zolang tot er een RESET of een interrupt optreedt.

Interrupt stuurinstructies

Een interrupt (onderbreking) heeft voornamelijk een taak bij belangrijke processen in de computer. Bij een interrupt deelt een bouwsteen de processor mee dat een bepaalde toestand is opgetreden (bijvoorbeeld: I/O-apparaat wacht op invoer). De CPU verwerkt deze mededelingen in volgorde van belangrijkheid. De interrupt onderbreekt een programma. Interrupts spelen bij in- en uitvoer een belangrijke rol. Met de MSX kunt u ook in BASIC interrupts programmeren (ON INTERVAL, ON STOP, etc.). De klok in de processor produceert bij deze instructies de interrupt. Wordt een toegestane interrupt opgeroepen, dan springt het programma naar het beginadres van een subroutine. RETI (return interrupt) herstelt het hoofdprogramma. We onderscheiden maskeerbare en niet-maskeerbare interrupts. De laatste worden altijd uitgevoerd en hebben de hoogste prioriteit. Met RETN springt u terug vanuit een nonmaskable interrupt.

DI (disable interrupt) en **EI** (enable interrupt)

DI zorgt ervoor dat geen rekening wordt gehouden met maskeerbare interrupts. De interrupts worden tegengehouden tot EI ze weer toelaat.

De Z80 kent drie interrupt-modi: IM 0, IM 1, en IM 2.

IM 0 (interrupt modus 0)

Met IM 0 schakelt u vanuit standaardmodus 1 naar modus 0. Na een interrupt wacht de processor in deze modus op een instructie van een extern apparaat.

IM 1

Dit is de standaardmodus na inschakeling van de computer. In deze modus wordt automatisch naar adres &H30 gesprongen.

IM 2 (vectorinterrupt)

In IM 2 vertakt u naar een adres in een tabel dat van een interrupt afhankelijk is.

Mnemonic	Symbolic Operation	Flags: C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
DAA	Converts acc. content into packed BCD following add or subtract with packed BCD operands	↕	↕	P	↕	•	↕	00 100 111	1	1	4	Decimal adjust accumulator
CPL	A ← $\bar{A}$	•	•	•	•	1	1	00 101 111	1	1	4	Complement accumulator (one's complement)
NEG	A ← 0 − A	↕	↕	V	↕	1	↕	11 101 101 01 000 100	2	2	8	Negate acc. (two's complement)
CCF	CY ← $\overline{CY}$	↕	•	•	•	0	X	00 111 111	1	1	4	Complement carry flag
SCF	CY ← 1	1	•	•	•	0	0	00 110 111	1	1	4	Set carry flag
NOP	No operation	•	•	•	•	•	•	00 000 000	1	1	4	
HALT	CPU halted	•	•	•	•	•	•	01 110 110	1	1	4	
DI	IFF ← 0	•	•	•	•	•	•	11 110 011	1	1	4	
EI	IFF ← 1	•	•	•	•	•	•	11 111 011	1	1	4	
IM 0	Set interrupt mode 0	•	•	•	•	•	•	11 101 101 01 000 110	2	2	8	
IM 1	Set interrupt mode 1	•	•	•	•	•	•	11 101 101 01 010 110	2	2	8	
IM2	Set interrupt mode 2	•	•	•	•	•	•	11 101 101 01 011 110	2	2	8	

Notes: IFF indicates the interrupt enable flip-flop
CY indicates the carry flip-flop.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

4.13 In- en uitvoerinstructies

I/O-instructies zijn een verwaarloosde groep instructies van de Z80. Dat komt doordat werking en gebruikswijze van deze instructies afhankelijk zijn van de hardware. Elke computer heeft tenminste een I/O-bouwsteen voor randapparatuur. De MSX heeft een PPI (Programmable Peripheral Interface (8255)), ook wel aangeduid als PIO of PIA. Een aantal I/O-instructies spreekt deze bouwsteen aan, die vervolgens zelfstandig communiceert met toetsenbord, cassette en geheugenbeheer. Via I/O-lijnen zijn met de processor ook verbonden:

- de PSG (programmable sound generator); zorgt voor de produktie van geluid en joystick-uitlezing
- de VDP (video display processor); bestuurt het beeldscherm

Alle bouwstenen die met I/O-instructies worden aangesproken, zijn belangrijke onderdelen van de computer. De I/O-instructies vormen de verbinding tussen deze zeer zelfstandig opererende bouwstenen (de VDP bijvoorbeeld regelt onafhankelijk van de CPU het video-uitgangssignaal). Ze regelen de wederzijdse overdracht van gegevens en instructies van en naar de I/O-bouwstenen. Die zorgen dan voor de feitelijke communicatie met de externe apparaten. In computertaal noemen we deze verbindingen meestal port of interface. Een I/O-instructie schrijft simpelweg een waarde naar de betreffende poort. Als het bijbehorende apparaat is aangesloten, wordt de waarde ontvangen en de gewenste handeling uitgevoerd. Normaal gesproken zijn er 256 I/O-adressen mogelijk. Het adres bepaalt naar welk apparaat de data moeten worden gestuurd. Belangrijke I/O-poorten komen we verderop nog tegen. Uit het bovenstaande volgt dat voor een complete instructie twee gegevens nodig zijn:

- het adres van de I/O-poort
- de waarden van de gegevens die moeten worden verzonden, of de aanduiding van het register waar de ontvangen waarden heen moeten

Het adres van de poort moet u altijd tussen haakjes zetten. De instructie IN transporteert gegevens van de poort naar het aangegeven register; de instructie OUT stuurt gegevens naar de actuele poort. Er zijn twee manieren om I/O-instructies te adresseren:

Onmiddellijke adressering

formaat OUT (n),A IN A,(n)

Indirecte adressering

formaat OUT (C),r IN r,(C)

MSX-BASIC kent de instructies INP en OUT. De werking ervan is gelijk aan die van machinetaalinstructies, maar veel toepassingen zijn alleen in machinetaal uitvoerbaar. Daarnaast zijn er nog vier instructies voor blokin- en uitvoer. U kunt ze op dezelfde manier gebruiken als bloktransfer-instructies: HL wijst op de actuele geheugenplaats, C is het adres van de betreffende poort en B is de lengte van het blok. I/O-instructies beginnen met de code &HED en hebben een 2-bytes opcode. Alleen de direct geadresseerde instructies IN A,(n) en OUT (n),A hebben een 1-bytes opcode.

Mnemonic	Symbolic Operation	Flags C	Z	P/V	S	N	H	Op-Code 76 543 210	No. of Bytes	No. of M Cycles	No. of T States	Comments
IN A, (n)	A ← (n)	•	•	•	•	•	•	11 011 011 ← n →	2	3	11	n to $A_0 \sim A_7$ Acc to $A_8 \sim A_{15}$
IN r, (C)	r ← (C) if r = 110 only the flags will be affected	•	↕	P	↕	0	↕	11 101 101 01 r 000	2	3	12	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INI	(HL) ← (C) B ← B - 1 HL ← HL + 1	X	① ↕	X	X	1	X	11 101 101 10 100 010	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INIR	(HL) ← (C) B ← B - 1 HL ← HL + 1 Repeat until B = 0	X	1	X	X	1	X	11 101 101 10 110 010	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
IND	(HL) ← (C) B ← B - 1 HL ← HL - 1	X	① ↕	X	X	1	X	11 101 101 10 101 010	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
INDR	(HL) ← (C) B ← B - 1 HL ← HL - 1 Repeat until B = 0	X	1	X	X	1	X	11 101 101 10 111 010	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OUT (n), A	(n) ← A	•	•	•	•	•	•	11 010 011 ← n →	2	3	11	n to $A_0 \sim A_7$ Acc to $A_8 \sim A_{15}$
OUT (C), r	(C) ← r	•	•	•	•	•	•	11 101 101 01 r 001	2	3	12	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OUTI	(C) ← (HL) B ← B - 1 HL ← HL + 1	X	① ↕	X	X	1	X	11 101 101 10 100 011	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OTIR	(C) ← (HL) B ← B - 1 HL ← HL + 1 Repeat until B = 0	X	1	X	X	1	X	11 101 101 10 110 011	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OUTD	(C) ← (HL) B ← B - 1 HL ← HL - 1	X	① ↕	X	X	1	X	11 101 101 10 101 011	2	4	16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$
OTDR	(C) ← (HL) B ← B - 1 HL ← HL - 1 Repeat until B = 0	X	1	X	X	1	X	11 101 101 10 111 011	2 2	5 (If B ≠ 0) 4 (If B = 0)	21 16	C to $A_0 \sim A_7$ B to $A_8 \sim A_{15}$

Notes: ① If the result of B - 1 is zero the Z flag is set, otherwise it is reset.

Flag Notation: • = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown,
↕ = flag is affected according to the result of the operation.

Toepassing

Om de I/O-instructies te demonstreren, passen we ze toe op de produktie van geluid. U kent natuurlijk de klik als u een toets op de MSX aanraakt. Dit geluid wordt bij wijze van uitzondering niet voortgebracht door de PSG (de geluids-chip). Wel bestaat de mogelijkheid via de PPI een klik op de audio-uitgang te krijgen. Dat gaat simpelweg door een lijn die met die uitgang verbonden is te zetten of te resetten. De omschakeling van de ene toestand naar de andere veroorzaakt een beweging van de luidsprekerconus die als een klik klinkt. Het aardige is dat het niet bij een klik hoeft te blijven; als de omschakeling zeer vaak en snel achter elkaar plaatsvindt, klinkt dat als een toon. Schakel eerst de toetsklik uit met SCREEN,,0 of met POKE &HF3DB,0. Voer vervolgens in:

```
OUT &HAB, 15
```

Hoewel de klik nu is uitgeschakeld, produceert de RETURN-toets dat geluid nog steeds. Dit geluid wijst erop dat de luidsprekerconus gespannen staat. Herhaalt u deze instructie, dan hoort u niets omdat de toestand van de conus niet verandert. Met

```
OUT &HAB, 14
```

wordt de luidsprekerspanning op 0 gezet. Dat veroorzaakt weer een klik. De actuele toestand van deze zogenaamde software sound-lijn komen we te weten door de instructie INP. Bit 7 van poort C geeft de toestand van de lijn aan. Het adres van die poort is &HAA. Om bit 7 afzonderlijk te bekijken gebruiken we de logische instructie

```
PRINT INP(&HAA) AND 2^7
```

Het resultaat 0 betekent dat de lijn op 0 staat, 128 geeft de toestand 1 aan. Het volgende programma zorgt ervoor dat de spanning honderd maal aan en uit geschakeld wordt.

```
10 FOR I=0 TO 100
20 OUT &HAB, 15
30 OUT &HAB, 14
40 NEXT
```

Door een regel 25 in te voegen met een wachtlus, een REM- of een PRINT-instructie, kunt u de snelheid waarmee het programma wordt uitgevoerd vertragen. Weinig trillingen, dus een lage frequentie,

betekent een lage toon, veel trillingen, dus een hoge frequentie een hoge toon. In BASIC is deze manier van geluid maken niet zo zinvol omdat het programma niet snel genoeg is om boven de 250 Hz te komen. Wilt u dat toch, dan hebt u machinetaal nodig. Om geluid te produceren schrijven we dus een machinetaalprogramma. De opbouw van het programma is gelijk aan dat in BASIC. De buitenste lus regelt de duur van de toon. Binnen deze lus ligt een lus die ervoor zorgt dat de spanning in- en uitgeschakeld wordt. Een wachtlus die hier weer binnen ligt, bepaalt het tempo van de schakelingen en daarmee de frequentie. Net als in het BASIC-voorbeeld zorgt de instructie OUT (C),r voor de klik.

```
10 '                  ; blokpuls
20 ' LD C,&HAB        ; poortadres
30 ' LD B,&HF         ; geluid op high
40 ' LD DE,(LENGTE)   ; toonlengte
50 ' TOON OUT (C),B   ; lijn op 1
60 ' DEC DE           ; teller verlagen
70 ' LD HL,(FREWA)    ; teller van wachtlus
                      ; die frequentie bepaalt
80 ' WACH1 DEC HL     ; teller verlagen
90 ' LD A,H           ; is teller 0?
100 ' OR L
110 ' JR NZ,WACH1     ; teller <> 0, dan wachten
120 ' DEC B           ; geluid op low
130 ' OUT (C),B       ; lijn op 0
140 ' LD HL,(FREWA)   ; teller
150 ' WACH2 DEC HL    ; tweede
160 ' LD A,H          ; wachtlus
170 ' OR L
180 ' JR NZ,WACH2
190 ' INC B           ; geluid op high
200 ' LD A,D          ; is teller
210 ' OR E            ; voor lengte 0?
220 ' JR NZ,TOON      ; teller <> 0, dan volgende
                      ; trilling produceren
230 ' RET             ; teller=0, dan terug
240 ' LENGTE DW 440   ; toonduur = 1 sec
250 ' FREWA DW 155    ; frequentie = 440 Hz
```

Om te begrijpen hoe de waarden van de lengtelus en de frequentiewachtlus tot stand komen, moeten we iets dieper op de zaak ingaan. De Z80 rafelt machinetaalinstructies uiteen in elementaire operaties. Elke instructie bestaat uit een exact vastgelegde serie elementaire handelingen of machinecycli. Een instructie bestaat uit 1-5 cycli (M1-M5) en iedere cyclus duurt 3-6 perioden. De duur van een periode wordt bepaald door de klokfrequentie van de CPU. De klok is een zeer belangrijk

onderdeel van de computer omdat deze ervoor zorgt dat alle processen ondanks de zeer hoge snelheden synchroon verlopen. De MSX-norm voor de klokfrequentie is 3.58 MHz. De periode, oftewel een trilling van de klok, duurt derhalve

$$\frac{1}{3.58\ \text{MHz}} = \frac{1}{3.580.000\ \text{Hz}} = 0.00000028\ \text{sec} = 0.28\ \text{microsec.}$$

Deze tijdsduur noemen we de klokcyclus, T (van time). We gaan nu stap voor stap na hoe een instructie wordt uitgevoerd. De M1-cyclus heet in het Engels instruction fetch cycle (to fetch = ophalen). Deze cyclus duurt minstens 4 klokcycli en verloopt als volgt.

M1 instruction fetch cycle

T1	PC uitvoeren op adresbus
T2	PC verhogen
T3	waarde op databus naar buffer; dit is de waarde op het adres waar PC bij T1 naar wees
T4,5,6	uitvoering van handelingen die afhankelijk zijn van de instructie

Alle verder benodigde M-cycli beslaan tussen de drie en vijf T-cycli. Op grond van het aantal benodigde T-cycli kunnen we precies berekenen hoe lang het duurt om een instructie uit te voeren. De kortste instructies bestaan uit een M1-serie van vier cycli (bijvoorbeeld LD r,r'). Bij een klokfrequentie van 3.58 MHz duurt dat 4 * 0.28 microsec. = 1.12 microsec. Dat heeft tot gevolg dat per seconde bijna 1 miljoen van zulke instructies kunnen worden uitgevoerd. In de instructielijsten staat bij elke instructie het benodigde aantal M- en T-cycli aangegeven (No. of M cycles en No. of T States).

Opgave

Reken uit hoe lang het duurt om de wachtlus die de frequentie bepaalt 155 keer te doorlopen.

Oplossing

```
instructie                aantal T-cycli

WACHT DEC HL               6
      LD  A,H              4
      OR  L                4
      JR  NZ,WACHT        12 (7)
                          ---
                          26
```

Totale duur:

```
(155 * 26 - 5) * 0.28 microsec. = 11.127 microsec.
                                =  1.127 millisec.
```

Voor een complete trilling moet de wachtlus tweemaal worden doorlopen. De trillingsduur van de geproduceerde toon is ongeveer 1.127 + 1.127 = 2.254 millisec. De frequentie is de omgekeerde waarde van de trillingsduur:

$$\frac{1}{2.254 \text{ msec.}} = \frac{1}{0.002254 \text{ sec.}} = 443.7 \text{ Hz}$$

Dit is een benadering omdat we geen rekening hebben gehouden met de instructies buiten de lussen:

```
1                           2

DEC DE            6         LD  HL,(FREWA)     16
LD  HL,(FREWA)   16         INC B               4
DEC B             4         LD  A,D             4
OUT (C),B        12         OR  E               4
                            JR  NZ,TOON        12
                            OUT (C),B          12
                 --                            --
                 38                            52
```

D1 = (38 + N1 * 26 - 5) * T = 26 * N1 + 33 * T

D2 = (52 + N2 * 26 - 5) * T = 26 * N2 + 47 * T

$$T: \text{duur van een klokcyclus} = \frac{1}{3.579545} \text{ microsec.}$$

N: het aantal keren dat een lus wordt doorlopen

Voor de precieze berekening van T hebben we een nauwkeuriger waarde van de klokfrequentie gebruikt: 3.579545 MHz. De totale trillingsduur D is de som van D1 en D2. D is het omgekeerde van de frequentie F.

$$1 / F = 26 * T * (N1 + N2) + 80 * T$$

anders geschreven:

$$N1 + N2 = \frac{1 / F - 80 * T}{26 * T} = \frac{1}{F * 26 * T} - \frac{80}{26}$$

$$T = \frac{1}{3.579545} \text{ microsec.} = \frac{1}{3579545}$$

gesubstitueerd in de vorige regel, resulteert in

$$N1 + N2 = \frac{3579545}{F * 26} - 3.08$$

Als we een frequentie van 440 Hz willen hebben, vinden we

$$N1 + N2 = 3579545 / (440 * 26) - 3.08 = 309.8$$

In het programma maken we geen onderscheid tussen N1 en N2 (beide heten FREWA). Conclusie: N1 = N2 = FREWA = 309.8 / 2 = 155. Bij deze waarde van FREWA wordt de buitenlus (met teller DE) 440 maal doorlopen (= de frequentie). Hierdoor wordt een toon van precies 1 seconde geproduceerd, en dat is ook het juiste antwoord op de vraag.

De toon klinkt wat merkwaardig als u het programma test. Met regelmatige tussenpozen wordt de toon onderbroken en daarom klinkt hij niet als de normale muziektoon van 440 Hz, de A. In het programma zelf is geen sprake van onderbrekingen. De oplossing van dit probleem is de interrupt. Elk programma wordt om de 50e seconde onderbroken door een interrupt. Het programma vertakt naar een bepaalde plaats in het bedrijfssysteem, waar wordt gecontroleerd of er bijvoorbeeld is gedrukt op een toets (STOP) of

op de vuurknop van de joy-stick (ON STRIG). De inwendige klok loopt dan ook een tik verder (om de tijd in seconden te weten te komen, moeten we daarom de variabele TIME door 50 delen). Het in dit geval hinderlijke effect van de interrupts kunnen we uitschakelen met DI (disable interrupt). Met EI (enable interrupt) laten we ze weer toe. Zet deze instructies respectievelijk in regel 45 en 225. We veranderen het programma zo dat de wachtlussen onafhankelijk van elkaar lopen:

```
 70 ' LD HL,(FREWA1)
140 ' LD HL,(FREWA2)
240 ' LENGTE DW 400
250 ' FREWA1 DW 172
255 ' FREWA2 DW 172
```

Na invoer van RENUM ziet de complete assemblerlisting er als volgt uit:

```
     F000            10                ; blokpuls
     F000 0EAB       20           LD   C,&HAB
     F002 060F       30           LD   B,&HF
    ?F004 ED5B0000   40           LD   DE,(LENGTE)
     F008 F3         50           DI
     F009 ED41       60   TOON    OUT  (C),B
     F00B 1B         70           DEC  DE
    ?F00C 2A0000     80           LD   HL,(FREWA1)
     F00F 2B         90   WACH1   DEC  HL
     F010 7C        100           LD   A,H
     F011 B5        110           OR   L
     F012 20FB      120           JR   NZ,WACH1
     F014 05        130           DEC  B
     F015 ED41      140           OUT  (C),B
    ?F017 2A0000    150           LD   HL,(FREWA2)
     F01A 2B        160   WACH2   DEC  HL
     F01B 7C        170           LD   A,H
     F01C B5        180           OR   L
     F01D 20FB      190           JR   NZ,WACH2
     F01F 04        200           INC  B
     F020 7A        210           LD   A,D
     F021 B3        220           OR   E
     F022 20E5      230           JR   NZ,TOON
     F024 FB        240           EI
     F025 C9        250           RET
**** regel  40  : LENGTE=&HF026
     F026 9001       260  LENGTE DW   400
**** regel  80  : FREWA1=&HF028
     F028 AC00       270  FREWA1 DW   172
**** regel  150  : FREWA2=&HF02A
     F02A AC00       280  FREWA2 DW   172
```

```
-------------------------------------------------
programma: Toon
begin: &HF000   einde: &HF02B
lengte: &H2C bytes
fout:  0
variabelen:
TOON   F009   WACH1  F00F
WACH2  F01A   LENGTE F026
FREWA1 F028   FREWA2 F02A
```

Regel 40 heeft betrekking op het label LENGTE dat pas in regel 260 weer voorkomt. Daarom wordt eerst code ED5B0000 opgeborgen. Bij de vertaling van regel 260 vindt de assembler het label LENGTE en meldt dat het voorkomt in regel 40. Code ED5B0000 wordt daar automatisch verbeterd. Dit kan ook gebeuren bij JR, CALL en JP als in het programma een voorwaartse sprong optreedt. We moeten sprongen vooruit op deze manier behandelen omdat de assembler het source-programma maar 1 keer doorloopt. Een twee-fasen-assembler (Engels: two-pass assembler) zoekt eerst alle variabelen en labels op en geeft ze een waarde. Pas bij de tweede doorloop begint de vertaling. Grote, professionele assembleerprogramma's werken met nog meer fasen. In ons geval is een een-fasen-assembler erg handig omdat hij ongeveer tweemaal zo snel is als een twee-fasen-type.

Om de besproken routine gemakkelijk te kunnen gebruiken, heeft u nog een BASIC-programma nodig dat na invoer van frequentie, toonduur en verhouding high/low de waarden uitrekent, nodig voor het machinetaalprogramma, en ze met POKE naar de betreffende plaatsen stuurt. De high/low-verhouding geeft de verhouding aan tussen de beide wachtluswaarden van het assemblerprogramma (N1 (lowpuls) en N2 (highpuls)). Zolang de som van deze waarden constant blijft, behoudt de toon dezelfde frequentie. Door een verandering van de verhouding krijgt u een andere klankkleur en geluidssterkte.

BASIC bedieningsprogramma

```
10 REM blokpuls
20 CLEAR 20,&HEFFF
30 CLS
40 INPUT"frequentie ";F
50 INPUT"duur";D
60 INPUT"hi/lo";P
70 L=D*F
80 REM lengte (aantal cycli)
```

```
90 POKE &HF026,L-INT(L/256)*256
100 POKE &HF027,INT(L/256)
110 W=3579545#/2/F/26-80/26
120 W2=W/(1+P)
130 W1=W2*P
140 REM frequentie
150 POKE &HF028,W1-INT(W1/256)*256
160 POKE &HF029,INT(W1/256)
170 POKE &HF02A,W2-INT(W2/256)*256
180 POKE &HF02B,INT(W2/256)
190 X=USR1(1)
200 GOTO 40
```

5.1 De disassembler

De MSX heeft 32 Kbyte ROM, vol met systeemroutines. De bovenste 16 Kbyte ROM (&H4000-&H7FFF) bevatten het BASIC, de onderste 16K (&H0-&3HFFF) het bedrijfssysteem van de computer. In het bedrijfssysteem staan interessante routines voor programmeerprojecten in machinetaal. Met een disassembler kunnen we die routines analyseren. Een disassembler interpreteert de bytes van een ingevoerd bereik als machinetaal en vertaalt de getallen in assemblerinstructies. De disassembler is dus de tegenhanger van de assembler. De hexadecimale getallen die met een BASIC-lader (in DATA-regels) zijn ingevoerd, kunnen met een disassembler worden terugvertaald in assemblerinstructies. Ook ROM-routines kunnen worden vertaald. Deze professionele programma's zijn bijzonder leerzaam en bovendien bruikbaar voor eigen programma's. Start de disassembler met RUN en voer vervolgens als beginadres &H146A in en als eindadres &H146F. Op het scherm ziet u:

```
146A 7C        LD   A,H
146B 92        SUB  D
146C C0        RET  NZ
146D 7D        LD   A,L
146E 93        SUB  E
146F C9        RET
```

Deze systeemroutine vergelijkt de inhoud van de registers HL en DE. Hoewel de Z80 er geen directe instructie voor heeft, komt deze opdracht vaak voor en daarom is er een RESTART-routine voor gemaakt. Die kunt u met een RST-instructie aanroepen. Bekijk adres &H20 (= sprongadres van de instructie RST &H20):

```
0020 C36A14    JP   &H146A
```

Ga na hoe de flags worden beïnvloed als HL > DE, HL < DE en HL = DE.

Nadat het opgegeven bereik is gedisassembleerd, wacht het programma tot een toets is ingedrukt. Met e of E beëindigt u het programma, bij alle andere toetsen springt het programma terug naar het beginmenu.

5.1.1 De listing van de disassembler

Kijk bij het intoetsen van de disassembler goed naar de programmabeschrijving.

```
10 CLEAR 200,&HF000:MAXFILES=0
20 SCREEN 0:COLOR 1,14,14:KEY OFF
30 GOTO 990
40 LOCATE 1,4:PRINT"Z 8 0 - D I S A S S E M B L E R"
50 E$="":LOCATE 3,7:INPUT"Printer (j/n) ";E$
60 LOCATE 3,10:INPUT"Startadres :&H";A$
70 GOSUB 900:AN=A
80 LOCATE 3,12:INPUT"Eindadres :&H";A$
90 GOSUB 900:EN=A
100 IF AN>EN THEN CLS:GOTO 40
110 IF E$="j" OR E$="J" THEN POKE &HFEE4,&HC3
120 PC=AN
130 CLS
140 AD=PC
150 PRINTRIGHT$("000"+HEX$(AD),4);"  ";
160 IL=0
170 GOSUB 940
180 GOSUB 300
190 IF IL THEN 600
200 IF LEFT$(PR$,3)="RST" AND (W=&HCF) THEN PR$=PR$+" /DB:n"
210 IF INSTR(PR$,"n")<>0 THEN 700
220 IF INSTR(PR$,"e")<>0 THEN 820
230 PO=INSTR(PR$," ")
240 IF PR$="" THEN PR$="???"
250 IF PO=0 THEN PRINTTAB(20);PR$;:GOTO 270
260 PRINTTAB(20);LEFT$(PR$,PO-1);TAB(25);RIGHT$(PR$,LEN(PR$)-PO);
270 PRINT
280 IF PC<=EN THEN 140 ELSE POKE &HFEE4,&HC9
285 A$=INKEY$:IF A$="" THEN 285
290 IF A$<>"E" AND A$<>"e" THEN CLS:GOTO 40
295 POKE &HFEE4,&HC9:KEY ON:END
300 REM interpretatie ----------
310 IF (W=&HDD OR W=&HFD) AND NOT IL THEN 490
320 IF W=&HED THEN 460
330 IF W=&HCB THEN 410
340 GOSUB 540
350 ON C1 GOTO 370,390,360
360 PR$=BF$(W):RETURN
370 IF W=&H76 THEN PR$="HALT":RETURN
```

```
380 PR$="LD "+RT$(C2)+","+RG$:RETURN
390 IF C2=0 OR C2=1 OR C2=3 THEN A$=" A," ELSE A$=" "
400 PR$=AL$(C2)+A$+RG$:RETURN
410 REM instructies met als eerste byte &HBC ----------
420 GOSUB 940
425 IF IL THEN DI=W:GOSUB 940
430 GOSUB 540
440 IF C1=0 THEN PR$=RS$(C2)+" "+RG$ ELSE
 PR$=BI$(C1)+STR$(C2)+","+RG$
450 RETURN
460 REM instructies met als eerste byte &HED ----------
470 GOSUB 940
480 IF W<&H40 OR W>&HBF THEN PR$="???":RETURN ELSE GOTO 360
490 REM geindiceerde instructies ----------
500 IL=-1
510 IF W=&HDD THEN I$="IX" ELSE I$="IY"
520 GOSUB 940
530 GOTO 300
540 REM code opsplitsen ----------
550 C1=(W AND &B11000000)/64
560 C2=(W AND &B111000)/8
570 C3=W AND &B111
580 RG$=RT$(C3)
590 RETURN
600 REM geindiceerd ----------
610 PO=INSTR(PR$,"HL")
620 IF PO=0 THEN PR$="???":GOTO 230
630 IF INSTR(PR$,"(HL)")<>0 THEN 670
640 IF PR$="EX DE,HL" THEN PR$="???":GOTO 230
650 IF PR$="ADD HL,HL" THEN PR$="ADD "+I$+","+I$:GOTO 230
660 PR$=LEFT$(PR$,PO-1)+I$+RIGHT$(PR$,LEN(PR$)-PO-1):GOTO 200
670 IF LEFT$(PR$,2)="JP" THEN 660
680 IF PC-ADR<3 THEN GOSUB 940:DI=W
685 IF DI>127 THEN DI$=STR$(DI-256) ELSE
 DI$="+"+RIGHT$(STR$(DI),LEN(STR$(DI))-1)
690 I$=I$+DI$:GOTO 660
700 REM n vervangen ----------
710 PO=INSTR(PR$,"nn")
720 IF PO<>0 THEN 770
730 PO=INSTR(PR$,"n")
740 GOSUB 940
750 PR$=LEFT$(PR$,PO-1)+"&H"+RIGHT$("00"+HEX$(W),2)
+RIGHT$(PR$,LEN(PR$)-PO)
760 GOTO 230
770 GOSUB 940:LB=W
780 GOSUB 940
790 WE=W*256+LB
800 PR$=LEFT$(PR$,PO-1)+"&H"+RIGHT$("0000"+HEX$(WE),4)
```

```
+RIGHT$(PR$,LEN(PR$)-PO-1)
810 GOTO 230
820 REM e vervangen ----------
830 PO=INSTR(PR$,"e")
840 GOSUB 940
850 IF W>127 THEN W=W-256:REM twee complement
860 W=W+2
870 A$="$"+STR$(W)+" >"+"&H"+RIGHT$("0000"+HEX$(PC+W-2),4)
880 PR$=LEFT$(PR$,PO-1)+A$+RIGHT$(PR$,LEN(PR$)-PO)
890 GOTO 230
900 REM omzettig hex -> dec ----------
910 IF A$="" THEN A=0:RETURN
920 A=VAL("&H"+A$)
930 RETURN
940 REM byte lezen ----------
950 W=PEEK(PC)
960 PC=PC+1
970 PRINTRIGHT$("00"+HEX$(W),2);" ";
980 RETURN
990 REM variabelen initialiseren ----------
1000 DIM RT$(7),RS$(8),BI$(3),AL$(7),BF$(255)
1010 FOR I=0 TO 7:READ RT$(I):NEXT
1020 FOR I=0 TO 7:READ RS$(I):NEXT
1030 FOR I=1 TO 3:READ BI$(I):NEXT
1040 FOR I=0 TO 7:READ AL$(I):NEXT
1050 FOR I=0 TO &H7F:READ BF$(I):NEXT
1060 FOR I=&H80 TO &H9F:BF$(I)="":NEXT
1070 FOR I=&HA0 TO &HFF:READ BF$(I):NEXT
1080 GOTO 1400
1090 REM instructietabellen ----------
1100 DATA B,C,D,E,H,L,(HL),A
1110 DATA RLC,RRC,RL,RR,SLA,SRA,???,SRL
1120 DATA BIT,RES,SET
1130 DATA ADD,ADC,SUB,SBC,AND,XOR,OR,CP
1140 DATA NOP,"LD BC,nn","LD (BC),A",INC BC,INC B,
DEC B,"LD B,n",RLCA
1150 DATA "EX AF,AF'","ADD HL,BC","LD A,BC",DEC BC,
INC C,DEC C,"LD C,n",RRCA
1160 DATA DJNZ e,"LD DE,nn","LD (DE),A",INC DE,INC D,
DEC D,"LD D,n",RLA
1170 DATA JR e,"ADD HL,DE","LD A,(DE)",DEC DE,INC E,
DEC E,"LD E,n",RRA
1180 DATA "JR NZ,e","LD HL,nn","LD (nn),HL",INC HL,
INC H,DEC H,"LD H,n",DAA
1190 DATA "JR Z,e","ADD HL,HL","LD HL,(nn)",DEC HL,
INC L,DEC L,"LD L,n",CPL
1200 DATA "JR NC,e","LD SP,nn","LD (nn),A",INC SP,
INC (HL),DEC (HL),"LD (HL),n",SCF
```

```
1210 DATA "JR C,e","ADD HL,SP","LD A,(nn)",DEC SP,
INC A,DEC A,"LD A,n",CCF
1220 DATA "IN B,(C)","OUT (C),B","SBC HL,BC","LD (nn),BC",
NEG,RETN,IM 0,"LD I,A"
1230 DATA "IN C,(C)","OUT (C),C","ADC HL,BC","LD BC,(nn)",
,RETI,,"LD R,A"
1240 DATA "IN D,(C)","OUT (C),D","SBC HL,DE","LD (nn),DE",
,,IM 1,"LD A,I"
1250 DATA "IN E,(C)","OUT (C),E","ADC HL,DE","LD DE,(nn)",
,,IM 2,"LD A,R"
1260 DATA "IN H,(C)","OUT (C),H","SBC HL,HL","LD (nn),HL",
,,,RRD
1270 DATA "IN L,(C)","OUT (C),L","ADC HL,HL","LD HL,(nn)",
,,,RLD
1280 DATA ,,"SBC HL,SP","LD (nn),SP",,,,
1290 DATA "IN A,(C)","OUT (C),A","ADC HL,SP","LD SP,(nn)",,,,
1300 DATA LDI,CPI,INI,OUTI,,,,,LDD,CPD,IND,OUTD,,,,
1310 DATA LDIR,CPIR,INIR,OTIR,,,,,LDDR,CPDR,INDR,OTDR,,,,
1320 DATA RET NZ,POP BC,"JP NZ,nn",JP nn,"CALL NZ,nn",
PUSH BC,"ADD A,n",RST &H00
1330 DATA RET Z,RET,"JP Z,nn",->,"CALL Z,nn",CALL nn,
"ADC A,n",RST &H08
1340 DATA RET NC,POP DE,"JP NC,nn","OUT (n),A",
"CALL NC,nn",PUSH DE,"SUB n",RST &H10
1350 DATA RET C,EXX,"JP C,nn","IN A,(n)","CALL C,nn",->,
"SBC A,n",RST &H18
1360 DATA RET PO,POP HL,"JP PO,nn","EX (SP),HL","CALL PO,nn",
PUSH HL,"AND n",RST &H20
1370 DATA RET PE,JP (HL),"JP PE,nn","EX DE,HL","CALL PE,nn",
->,"XOR n",RST &H28
1380 DATA RET P,POP AF,"JP P,nn",DI,"CALL P,nn",PUSH AF,
"OR n",RST &H30
1390 DATA RET M,"LD SP,HL","JP M,nn",EI,"CALL M,nn",->,
"CP n",RST &H38
1400 ON STOP GOSUB 295
1410 STOP ON
1420 ON ERROR GOTO 295
1430 POKE &HFEE5,&H70:POKE &HFEE6,&HF3
1440 FOR I=&HF370 TO &HF37B:READ A:POKE I,A:NEXT
1450 GOTO 40
1460 DATA &HF5,&H3A,&H61,&HF6
1470 DATA &H32,&H15,&HF4,&HF1
1480 DATA &HCD,&H63,&H1B,&HC9
```

5.1.2 Programmabeschrijving

regel 10-130

Menu: invoer van begin- en eindadres. Keuze tussen printer en beeldscherm.

regel 140-290

Hier staat de hoofdlus van het programma.

regel 150

Deze regel voert het actuele adres uit.

regel 170

Volgende byte lezen en uitvoeren.

regel 180

Spring naar de subroutine die de interpretatie verricht.

regel 190

Spring naar de behandeling van geïndiceerde instructies als de flag IL is gezet (-1).

regel 200

Verwerk RST-instructies die de volgende data gebruiken.

regel 210

Vertak als de instructie getallen bevat.

regel 220

Vertak als de instructie relatieve afstanden bevat.

regel 230-270

Geformateerde uitvoer.

regel 280

Indien nog niet afgelopen, terug naar begin hoofdlus.

regel 295

Einde van het programma

regel 300-530

Bevat de subroutines voor de interpretatie.

regel 310

Spring naar de behandeling van geïndiceerde instructies.

regel 320

Spring naar de behandeling van instructies die met &HED beginnen.

regel 330

Spring naar de behandeling van instructies die met &HCB beginnen.

regel 340

Spring naar de subroutine die W splitst in C1 (bit6,7), C2 (bit 5-3) en C3 (bit 2-0).

regel 350

Bij C1=0 en C1=3 naar regel 360, dat wil zeggen: instructies uit de tabel lezen. Als C1=1 naar regel 370 (instructies in de vorm LD r,r'), als C1=2 naar regel 390 (8-bits logische of rekeninstructies).

regel 360

Bepaal PR$ uit de tabel.

regel 370-380

Instructies LD r,r' en HALT.

regel 390-400

Logische of rekeninstructies.

regel 410-450

Verwerk instructies die met &HCB beginnen.

regel 420

Lees volgende byte.

regel 425

Lees nog een byte bij een geïndiceerde instructie.

regel 430

Opsplitsen in C1, C2 en C3.

regel 440

PR$ produceren voor roteer- of schuifinstructies (C1=1) en bitmanipulatie-instructies.

regel 460-480

Verwerk instructies die met &HED beginnen.

regel 470

Lees volgende byte.

regel 480

Indien code geldig, zoek PR$ uit tabel (regel 360).

regel 490-530

Hier vindt de eerste behandeling plaats van de geïndiceerde instructies (uit regel 310).

regel 500

Zet de flag.

regel 510

Sla het register (IX of IY) op in I$.

regel 520

Lees volgende byte.

regel 530

Begin de interpretatie opnieuw (regel 300).

regel 540-590

Splits SUB-code op: W wordt ontleed in C1 (bit 7,6), C2 (bit 5-3) en C3 (bit 2-0). RG$ bevat het register dat bij C3 hoort.

regel 600-690

Tweede behandeling van geïndiceerde instructies (sprong vanaf regel 190). Test of de geïndiceerde instructie is toegestaan. Zo ja, dan wordt HL vervangen door I$. Als er een afstandsaanduiding nodig is, lezen de regels 680-690 de afstand.

regel 700-810

Substitueer een getal voor n als PR$ een n bevat.

regel 730-760

1-bytes getallen (n).

regel 770-810

2-bytes getallen (nn).

regel 820-890

Vervang offset (e).

regel 850-860:

Bereken offset.

regel 870-880

Offset bepalen.

regel 900-930

Subroutine die hexadecimale getallen verandert in decimale getallen.

regel 940-980

Subroutine: volgende byte lezen en uitvoeren.

regel 990-1080

Initialisering: opbouw van de tabellen.

regel 1090-1380

DATA-regels.

regel 1400-1480

Initialisering van de routines STOP en ERROR; uitvoerroutine inlezen, bestemd voor zowel printer als beeldscherm.

Variabelenlijst

A	resultaat van subroutine voor hex --> dec. Waarde van A$ als hex-getal geïnterpreteerd
A$	invoer van hex-getal
AD	adres van de eerste code van de actuele instructie
BEGA	vertaling beginadres
C1	bit 7 en 6
C2	bit 5-3
C3	bit 2-0
DIS	afstand (distance) bij geïndiceerde instructies
DIS$	afstandsstring voor uitvoer
EINA	eindadres vertalen
I$	actuele indexregister
IL	gezet bij geïndiceerde adressering, anders gereset
INV$	invoerstring (j/n)
LB	buffer met lowbyte: bij 2-bytes getallen
PC	programmateller: wijst op adressen van instructies
PO	positie van n, nn, e, HL... in PR$
PR$	print $, bevat assemblerinstructie
RG$	register: resultaat van subroutine die code splitst RG$ bevat het register dat bij C3 hoort
W	gelezen code
WAARDE	waarde van een 2-bytes getal (nn)

```
BI$(3)       bitmanipulatie-instructies
IN$(255)     0   -&H3F: instructies die met &HED beginnen en
                        als byte 1 het nummer van de
                        veldvariabelenteller bevatten
             &H40-&HBF: instructies die met &HED beginnen en
                        als byte 2 het nummer van de
                        veldvariabelenteller bevatten
             &HBF-&HFF: instructies die als byte 1 het
                        nummer van de veldvariabelenteller
                        bevatten
LR$(7)       logische of rekeninstructies
RS$(7)       roteer- en schuifinstructies
RT$(7)       register
```

5.2 Systeemroutines

Een van de belangrijkste systeemroutines zorgt ervoor dat er tekens op het scherm verschijnen. RST &H18 activeert deze routine. De routine voert het teken uit dat overeenkomt met de waarde in de accu. Schrijf een programma dat de karakterset van de MSX uitvoert (codes 32-255).

Oplossing

```
10 ' ORG &HF000
20 ' LD B,223              ; teller=255-32
30 ' LD A,32
40 ' LUS RST &H18          ; CHR$(A) uitvoeren
50 ' INC A
60 ' DJNZ LUS
70 ' RET
80 ' END
```

De pseudo-instructie DM van de assembler speelt bij de uitvoer van tekens een grote rol. Op DM volgt als operand een woord tussen aanhalingstekens. DM slaat de ASCII-code van de letters van dat woord op, te beginnen bij het actuele adres.

```
10 ' ORG &HF000
20 ' LD HL,WOORD    ; adres van het uit te voeren woord
30 ' LUS LD A,(HL)  ; accu laden met ASCII-code
                    ; van betreffende letter
40 ' INC HL         ; wijzer op volgende letter
50 ' RST &H18
60 ' OR A           ; flags zetten
70 ' JR NZ,LUS      ; ongelijk 0, dan volgende letter
80 ' RET
90 ' WOORD DM "COMPUTER"
100 ' DB 0
110 ' END
```

De door DB 0 geproduceerde nulbyte (regel 100) geeft het einde van het woord aan dat moet worden uitgevoerd. Het volgende onderwerp is een monitorprogramma, geschreven met de PRINT-routine.

5.2.1 De monitor

Behalve de assembler zijn er nog meer hulpmiddelen om het werken met machinetaal te vergemakkelijken. Een ervan is de monitor, waarmee natuurlijk niet het beeldscherm van de computer wordt bedoeld. Met een monitorprogramma kunnen we bijvoorbeeld de inhoud van het geheugen bekijken of veranderen (Engels: to monitor = controleren). Als de monitor gebruikersvriendelijk is, kunt u er ook machinetaalprogramma's mee laden, starten of in het geheugen opslaan. Door zo'n monitorprogramma te schrijven slaat u twee vliegen in een klap: enerzijds vergroot u uw kennis van programmeertechnieken en anderzijds is het resultaat een eenvoudig monitorprogramma.

Opgave

Zoals gezegd is de belangrijkste taak van een monitorprogramma de inhoud van het geheugen te laten zien (in BASIC bereikt u dat met de instructie PEEK). Schrijf een programma dat na invoer van een start- en een eindadres als uitvoer de tussenliggende geheugeninhoud geeft. Gebruik het normale formaat voor uitvoer van het geheugen in hexadecimale getallen (hex-dump genaamd). Het resultaat moet er zo uitzien:

```
0000 F3 C3 D7 02 BF 1B 98 98  sCW.?...
0008 C3 83 26 00 C3 B6 01 00  C.&.C6..
0010 C3 86 26 00 C3 D1 01 00  C.&.CQ..
0018 C3 45 1B 00 C3 17 02 00  CE..C...
0020 C3 6A 14 00 C3 5E 02 00  Cj..C^..
0028 C3 89 26 A1 13 00 00 00  C.&!....
0030 C3 05 02 00 00 00 00 00  C.......
```

Rechts naast de eigenlijke hex-dump staat de betekenis van de ASCII-codes. Codes groter dan 127 worden automatisch met 128 verminderd en codes zonder symbool (0-31) worden als punten uitgevoerd.

Oplossing

```
10 ON STOP GOSUB 170:STOP ON
20 ON KEY GOSUB 160:KEY (1) ON
30 KEY OFF:SCREEN 0
40 INPUT"start &h";A$
50 ST=VAL("&h"+A$)
60 INPUT"einde &h";A$
70 EI=VAL("&h"+A$)
80 FOR I=ST TO EI STEP 8:A$=""
90 PRINT RIGHT$("000"+HEX$(I),4);" ";
100 FOR J=0 TO 7:W=PEEK(I+J)
110 WP=W AND &H7F
120 IF WP<32 OR WP=127 THEN WP=46
130 A$=A$+CHR$(WP)
140 PRINT RIGHT$("0"+HEX$(W),2);" ";
```

```
150 NEXT:PRINT A$;:NEXT:PRINT:GOTO 40
160 J=7:I=EI:RETURN
170 KEY ON:CLS:END
```

Met dit programma kunt u in het ROM en het RAM kijken. Voer in het monitorprogramma de volgende regel in:

```
1    REM dit is de eerste regel
```

Als u naar de inhoud van &H8000-&H8030 kijkt, herkent u bovengenoemde eerste regel. Vanaf &H8000 worden immers de BASIC-programma's opgeslagen. Direct na het BASIC-programma volgt een intern beheerde pagina met alle in het programma gebruikte numerieke variabelen, waarvan de waarden direct worden opgeslagen, en de stringvariabelen, waarvan het adres en de lengte van de tekenreeks worden opgeslagen. De variabelen staan daar in de volgorde waarin ze in het programma voorkomen. We gaan nu een machinetaalprogramma ontwikkelen dat hetzelfde doet als het vorige, en nog meer. Om de geheugeninhoud uit te voeren in de vorm van hexadecimale getallen is een subroutine nodig. De bytes worden overgedragen aan de accu.

voorbeeld: A=63=&H3F=&B 0011 1111

3 bovenste 4 bits (lownibble)
F onderste 4 bits (highnibble)

Eerst wordt de highnibble uitgevoerd. Hiertoe verschuiven we de accu-inhoud 4 keer naar rechts (8-bits rotatie). Het resultaat is &B11110011. Daarna wissen we met AND de bovenste 4 bits. De accu heeft nu de waarde &B00000011=3 (ASCII-code 51). Deze waarde moet worden uitgevoerd. Om een accuwaarde van 51 te krijgen moet 48 bij de oorspronkelijke 3 worden opgeteld. Vervolgens wordt de PRINT-routine opgeroepen. Nu de lownibble. Eerst wissen we de bovenste 4 bits van de oude accu-inhoud. 48 erbij opgeteld levert 63 op. Het hexadecimale cijfer F (decimaal 15) moet worden uitgevoerd. De ASCII-waarde van F is 70. Dat betekent dat bij de waarde van de highnibble 7 moet worden opgeteld als het hexadecimale cijfer groter is dan 9, dus als het om een letter gaat. Pas daarna kan de PRINT-routine worden oproepen. Probeer eens de routine te schrijven die een byte uitvoert in hexadecimale vorm.

```
F000        10        ORG  &HF000
F000        30                        ; uithex
F000        40                        ; voert accu hex. uit
F000        50                        ; wijziging E-reg.
```

```
     F000 5F           60  UITHEX LD   E,A      ; accu bufferen
     F001 0F           70         RRCA          ; accu
     F002 0F           80         RRCA          ; 4 bits
     F003 0F           90         RRCA          ; naar rechts
     F004 0F          100         RRCA          ; verschuiven
     F005 E60F        110         AND  &B1111 ; bit 4-7 wissen
    ?F007 CD0000      120         CALL CONV     ; lownibble uitvoeren
     F00A 7B          130         LD   A,E      ; oude accu-inhoud
     F00B E60F        140         AND  &B1111 ; bit 4-7 wissen
    ?F00D CD0000      150         CALL CONV     ; lownibble uitvoeren
     F010 C9          160         RET           ; einde uithex
     F011             170                       ; CONV(ersie) routine
     F011             180                       ; voert accu hex. uit
**** regel  120  : CONV=&HF011
**** regel  150  : CONV=&HF011
     F011 FE0A        190  CONV   CP   &HA      ; waarde <10?
    ?F013 38FE        200         JR   C,GETAL; ja, dan naar getal
     F015 C607        210         ADD  A,7      ; +7 voor letters
**** regel  200  : GETAL=&HF017 offset 2
     F017 C630        220  GETAL  ADD  A,48     ; ASCII-code voor
                                                ; hex. getal
     F019 DF          230         RST  &H18
     F01A C9          240         RET
------------------------------------------------------------------------
programma:
begin: &HF000   einde: &HF01A
lengte: &H1B bytes
fout:  0
variabelen:
UITHEX F000   CONV   F011
GETAL  F017
```

We gaan verder met de monitorroutine. Vanuit BASIC worden de begin- en eindadressen overgedragen.

```
10 ' ORG &HF000
20 ' START DS 2
30 ' EINDE DS 2
```

Laad HL met het beginadres en voer het uit:

```
40 ' LD HL,(START)           ; wijzer
50 ' VER16 LD A,H            ; highbyte uitvoeren
60 ' CALL UITHEX
70 ' LD A,L                  ; lowbyte uitvoeren
80 ' CALL UITHEX
```

Daarna moet er een spatie worden uitgevoerd:

```
90 ' LD A,&H20                ; ASCII-code voor spatie
100 ' RST &H18
```

Nu voert het programma de waarde van de 8 volgende geheugenplaatsen uit:

```
110 ' LD B,8                  ; teller
120 ' VER LD A,(HL)           ; byte in accu laden
130 ' CALL UITHEX             ; en uitvoeren
140 ' LD A,&H20               ; spatie
150 ' CALL PRINT              ; uitvoeren
160 ' INC HL
170 ' DJNZ VER
```

Vervolgens voert het programma een spatie uit en de laatste 8 bytes als ASCII-tekens. Het trekt 128 af van codes groter dan 127 (bit 7 wordt gereset). Bij codes kleiner dan 32 (stuurtekens) verschijnen er punten (ASCII-waarde 46).

```
180 ' LD A,&H20
190 ' RST &H18                ; spatie
200 ' LD DE,8
210 ' OR A                    ; carry=0
220 ' SBC HL,DE               ; wijzer met 8 verlagen
230 ' LD B,8
240 ' VERAS LD A,(HL)         ; accu met byte laden
250 ' INC HL                  ; wijzer verhogen
260 ' RES 7,A                 ; ASCII-code van grafisch
                              ; teken
270 ' CP &H20                 ; >=32
280 ' JR NC,PR                ; ja, dan uitvoer
290 ' CP 127                  ; = DEL
300 ' JR NC,PR
310 ' LD A,46                 ; ASCII-code voor punt
320   PR RST &H18             ; teken uitvoeren
330   DJNZ VERAS
```

Om aan het begin van de volgende regel te komen, voeren we code 13 en code 10 in.

```
CHR$(13) = carriage return
CHR$(10) = line feed = volgende regel
```

```
340 ' LD A,13                 ; carriage return
350 ' RST &H18                ; uitvoeren
360 ' LD A,10                 ; line feed
370 ' RST &H18                ; uitvoeren
```

Vervolgens test het programma of het einde al is bereikt:

```
380 ' PUSH HL               ; wijzer op stack
390 ' LD DE,(EINDE)
400 ' OR A                  ; carry=0
410 ' SBC HL,DE             ; wijzer-einde <= 0
420 ' POP HL                ; wijzer halen; geen invloed op
                            ; flags
430 ' JR C,VER16            ; HL-DE<0, dan verder
440 ' JR Z,VER16            ; HL-DE=0, dan verder
450 ' RET                   ; HL-DE>0, dan einde
460 '                       ; einde monitor
```

Tot slot zetten we de UITHEX-routine voor of achter het programma. Daarna kunnen we van start.

```
470 '                       ; UITHEX
480 '                       ; voert accu hexadecimaal uit
490 '                       ; wijziging E-register
500 ' UITHEX LD E,A         ; accu bufferen
510 ' RRCA                  ; accu
520 ' RRCA                  ; 4 bits naar
530 ' RRCA                  ; rechts
540 ' RRCA                  ; schuiven
550 ' AND &B1111            ; bit 4-7 wissen
560 ' CALL CONV             ; highnibble uitvoeren
570 ' LD A,E                ; oude accu-inhoud
580 ' AND &B1111            ; bit 4-7 wissen
590 ' CALL CONV             ; lownibble uitvoeren
600 ' RET                   ; einde uithex
610 '                       ; CONV(ersion) routine
620 '                       ; voert accu hexadecimaal uit
630 ' CONV CP &HA           ; getal<10
640 ' JR C,GETAL            ; ja, dan naar getal
650 ' ADD A,7               ; +7 voor letters
660 ' GETAL ADD A,48        ; =ASCII-code van hexadecimaal
                            ; getal
670 ' RST &H18
680 ' RET                   ; einde CONV
690 ' END
```

5.3 De stap-voor-stap simulator

Programmeren in assembler is lastig omdat er geen eenvoudige manier is om de programma's te testen. Een fout komt in BASIC als syntax error te voorschijn maar leidt er in machinetaal meestal toe dat het systeem 'hangt'. Het is dus zinvol om een programma stap voor stap te kunnen doorlopen en de inhoud van registers op hun juistheid te controleren. Met de hand is dat een vervelend karwei, maar met een simulatieprogramma gaat het beter. Bij de start van het programma kunt u kiezen voor uitvoer via beeldscherm of printer. U kunt dat later nog wijzigen. Vervolgens begint de simulatie op adres 0000.

```
SZ H PNC A  B C  D E  H L  IX   IY

00000000 00 0000 0000 0000 0000 0000
00000000 00 0000 0000 0000 SP : F290
0000 F3        DI
```

Hier geeft de cursor aan dat het programma invoer verwacht. U heeft de keus uit:

ESC	programma beëindigen
SELECT	
RETURN	
"p" of "P"	uitvoer op printer
CAP	

Met RETURN verschijnt eventueel veranderde registerinhoud op het scherm, plus de volgende instructie:

```
SZ H PNC A  B C  D E  H L  IX   IY

00000000 00 0000 0000 0000 0000 0000
00000000 00 0000 0000 0000 SP : F290
0001 C3D702    JP   &H02D7
```

SELECT brengt het programma in de opmaakmodus zodat u naar believen de inhoud van de registers en de adressen van het programma kunt wijzigen. Daarbij moet u wel precies de hieronder besproken volgorde in acht nemen. Na SELECT staat de cursor automatisch onder S (voortekenflag) van de laatste registeruitvoer. Elke inhoud in de bovenste regel komt overeen met het actuele register. Binnen deze regel kunt u zich met de cursor of met CTRL-B en CTRL-F verplaatsen en op de aangegeven plaatsen de gewenste nieuwe inhoud invoeren in hexadecimale vorm.

Kleine letters en hoofdletters zijn toegestaan. De invoer voor het flagregister moet natuurlijk binair zijn. Dit register staat aan het begin van de regel (SZ H PNC) boven de actuele bits. De betekenis van de flags is:

S	voorteken (sign)
Z	nul (zero)
H	halve overdracht
P	P/V: pariteit of overflow
N	BCD-aftrekking
C	carry

Daarop volgen de accu A (8 bits) en de algemene registers B-L, eventueel samengevoegd tot paren van 16 bits. Aan het eind van de regel staan de registers IX en IY. Als u met de invoer in een regel klaar bent, komt u op de volgende regel door RETURN in te drukken. Dat mag niet met de cursor of andere toetsen gebeuren want daardoor zouden foute waarden kunnen worden geladen. Als u klaar bent met de veranderingen in de bovenste regel (actuele registers) kunt u hetzelfde doen in de tweede regel (schaduwregisters, op dit moment niet in gebruik). De opbouw van de tweede regel is analoog aan de eerste, maar de Z80 heeft maar 1 set indexregisters. Op die plaats staat daarom in de tweede regel de stackpointer SP. Let er goed op dat de SP tijdens de simulatie nooit kleiner wordt dan &HF259 want dan schrijft u over het simulatorprogramma in machinetaal heen. Gevolg: de computer 'hangt'. Wijzig zonodig de SP aan het begin van een simulatie omdat anders de volgorde van de terugsprongadressen in de war raakt. Sluit de regel weer af met RETURN. Daarna staat de cursor aan het begin van de regel met de vertaalde instructie. Daar kunt u de PC (eerste vier hex-tekens) veranderen. Het adres dat u hier invoert, geldt na een RETURN als adres waar de simulatie verder gaat. Hiermee verlaat u tevens de opmaakmodus. De laatste keuzemogelijkheid is het gebruik van CAP. Deze toets heeft hier niet zijn standaardfunctie en maakt in dit programma onderscheid tussen echte en pseudo simulatie. Echte simulatie heeft tot gevolg dat alle instructies daadwerkeljk worden uitgevoerd, wat bij sommige instructies fatale gevolgen kan hebben. Alle instructies die de configuratie van het systeem echt veranderen zijn 'gevaarlijk'. Door wijziging van een geheugeninhoud (bijvoorbeeld LD (&HF3DB),A) kunt u de gegevens veranderen die belangrijk zijn voor de BASIC-interpreter of het bedrijfssysteem. Ook I/O-instructies die echt worden gesimuleerd, kunnen de computer laten hangen omdat ze onder andere te maken hebben met de keuze van ROM of RAM. Bedachtzaamheid is ook geboden bij instructies die rechtstreeks op de SP slaan. U kunt daarom steeds voordat een instructie wordt uitgevoerd met CAP beslissen of dat daadwerkelijk moet gebeuren. Als de CAP-diode brandt, bent u

bezig met een echte simulatie. Zoniet, dan heeft een instructie geen invloed. Laten we als voorbeeld de instructie RST &H20 simuleren (zie de disassembler). Zet de PC op &H15BB en kies voor echte simulatie. Vergelijk uw resultaat met dat wat hieronder staat.

```
15BB 110001     LD   DE,&H0100
SZ H PNC A  B C  D E  H L  IX   IY
00000000 00 0000 0100 0000 0000 0000
00000000 00 0000 0000 0000 SP : F290
15BB E7         RST  &H20
SZ H PNC A  B C  D E  H L  IX   IY
00000000 00 0000 0100 0000 0000 0000
00000000 00 0000 0000 0000 SP : F28E
0020 C36A14     JP   &H146A
SZ H PNC A  B C  D E  H L  IX   IY
00000000 00 0000 0100 0000 0000 0000
00000000 00 0000 0000 0000 SP : F28E
146A 7C         LD   A,H
SZ H PNC A  B C  D E  H L  IX   IY
00000000 00 0000 0100 0000 0000 0000
00000000 00 0000 0000 0000 SP : F28E
146B 92         SUB D
SZ H PNC A  B C  D E  H L  IX   IY
10111011 FF 0000 0100 0000 0000 0000
00000000 00 0000 0000 0000 SP : F28E
146 C0          RET  NZ
SZ H PNC A  B C  D E  H L   IX   IY
10111011 FF 0000 0100 0000 0000 0000
00000000 00 0000 0000 0000 SP : F290
15BF 3804       JR   C,$6 >&H15C5
```

Toelichting

Eerst krijgt DE als inhoud &H100. Dan volgt een sprong naar RST &H20. Omdat het terugsprongadres (&H15BF) op de stack is gelegd, gaat de SP omlaag. JP &H146A brengt ons in de vergelijkingsroutine die de aftrekking HL-DE uitvoert. Omdat in dit geval DE>HL is, volgt een sprong terug met RET NZ. De gezette carry flag (=1) is een gevolg van het feit dat DE>HL is. Ga zelf nog eens na wat de werking van de simulator is vanaf adres &H15BB, maar nu met een andere inhoud van HL en DE (&H101, &H1000, &H80). Let op de gevolgen voor de flags. De instructie RET of RET NZ staat aan het eind van de routine. Nadat het terugsprongadres van de stack is gehaald, gaat het programma verder met de instructie die volgt op RST &H20. Als u programma's of instructies onduidelijk vindt, kunt u met de simulator nagaan hoe ze werken.

5.3.1 Beschrijving van SIMULA

Voordat we het programma systematisch toelichten eerst iets over de werking van de simulator. Het hoofdbestanddeel van de simulator is een machinetaalprogramma dat aan het einde van het programma wordt geladen. We hebben dit SIMULA genoemd. De uit te voeren instructie wordt als het ware midden in dit programma gePOKEt (regel 1850). Voordien zijn de registers al met de actuele waarden geladen. Na uitvoering van de instructie wordt elke eventueel veranderde registerinhoud opgeslagen om te worden omgezet in BASIC. De simulator roept dit programma aan met X=USR8(1) (regel 455). Voor een beter begrip hebben we de assemblerlisting van het machinetaalprogramma apart weergegeven. Ga aan de hand van de listing na hoe het programma werkt. Een beetje hocus pocus met de stack is daar wel voor nodig. Als u het wilt simuleren, moet u bij instructies die van invloed zijn op SP de echte simulatie uitschakelen en noodzakelijke veranderingen aanbrengen met SELECT. Om het programma zo kort mogelijk te houden, slaan we registers niet op met LD, maar lezen en schrijven we door middel van POP en PUSH. De SP wordt daarom van tevoren gezet op het begin van het bereik dat voor de overdracht is gereserveerd (SPOLA = SP overdracht low adres). Zo kunnen alle instructies worden uitgevoerd behalve spronginstructies. Bij een spronginstructie zouden we uit het machinetaalprogramma springen en daarna niet kunnen terugkeren naar BASIC. We behandelen de spronginstructies daarom geheel binnen BASIC, inclusief de bijbehorende stack- en PC-instructies. Denk er bij het intypen aan dat het laatste stuk van de simulator gelijk is aan de al eerder behandelde assembler, behoudens enkele kleine wijzigingen.

```
 F000           10                      ;simulator
 F200           20   ORG  &HF200
?F200 ED730000  30   LD   (SPREAL),SP   ;actuele SP op stack
?F204 310000    40   LD   SP,SPOLA      ;overdracht SP voor
                                        ;register
 F207 F1        50   POP  AF
 F208 C1        60   POP  BC
 F209 D1        70   POP  DE
 F20A E1        80   POP  HL
 F20B D9        90   EXX
 F20C 08        100  EX   AF,AF'
 F20D DDE1      110  POP  IX
```

```
 F20F FDE1       120     POP  IY
 F211 F1         130     POP  AF
 F212 C1         140     POP  BC
 F213 D1         150     POP  DE
 F214 E1         160     POP  HL
?F215 ED7B0000   170     LD   SP,(SPSIMU)   ;SP simulatie halen
 F219 0000       180  INSTRU DW  0          ;ruimte instructie
 F21B 0000       190          DW  0         ;ruimte instructie
?F21D ED730000   200     LD   (SPSIMU),SP   ;opslag SP simulatie
?F221 31000      210     LD   SP,(SPSIMU)
 F224 E5         220     PUSH HL
 F225 D5         240     PUSH DE
 F226 C5         250     PUSH BC
 F227 F5         260     PUSH AF
 F228 FDE5       270     PUSH IY
 F22A DDE5       280     PUSH IX
 F22C 08         290     EX   AF,AF'
 F22D D9         300     EXX
 F22E E5         310     PUSH HL
 F22F D5         320     PUSH DE
 F230 C5         330     PUSH BC
 F231 F5         340     PUSH AF
?F232 ED7B0000   350     LD   SP,(SPREAL)   ;actuele SP halen
 F236 C9         360     RET
 **** regel  30 : SPREAL=&HF237
 **** regel 350 : SPREAL=&HF237
 F237            370  SPREAL DS  2
 **** regel  40 : SPOLA=&HF239
 F239            380  SPOLA DS  20
 **** regel 170 : SPSIMU=&HF24D
 **** regel 200 : SPSIMU=&HF24D
 **** regel 210 : SPSIMU=&HF24D
?F24D 0000       390  SPSIMU DW  STACK
 F24F            400          DS  4         ;ruimte voor
                                            ;overflow sim.stack
 **** regel 390 : STACK=&HF253
 F253            410  STACK  DS  50         ;volgende plaats
                                            ;voor stack


 programma: SIMULA
 start: &HF200
 einde: &HF284
 lengte: &HF85
 0 fouten
 variabelentabel:
 INSTRU    F219
 SPOLA     F239
```

```
SPREAL    F237
SPSIMU    F24D
STACK     F253
```

5.3.2 De listing van de simulator

```
 10 CLEAR &H200,&HEFFF:MAXFILES=0
 20 SCREEN 0:COLOR 1,14,14:KEY OFF:WIDTH 37
 30 LOCATE 0,3:PRINT"Z 8 0   S I M U L A T O R - M S X"
 40 GOTO 1890
 50 LOCATE 3,7
 60 INPUT"printer (j/n)";E$:GOTO 460
 70 REM hoofdlus ----------
 80 IF E$="j" OR E$="J" THEN POKE &HFEE4,&HC3
 90 FOR I=S4 TO S4+3:POKE I,0:NEXT:QQ=0
100 GOSUB 1020:REM disassembleren ----------
110 POKE &HFEE4,&HC9:LOCATE ,,1
120 A$=INKEY$:IF A$="" THEN 120 ELSE LOCATE ,,0
130 IF A$=CHR$(27) THEN 2520:REM ESC = einde ----------
140 IF A$=CHR$(24) THEN 570:REM SELECT = veranderen ----------
150 IF A$="P" OR A$="p" THEN 60
160 IF A$<>CHR$(13) THEN 120:REM RETURN = doorgaan ----------
170 IF PEEK(&HFCAB)<>255 THEN 460:
REM echte simulatie alleen als CAP aan is ----------
180 IF PO=0 THEN B$=PR$:GOTO 210
190 B$=LEFT$(LEFT$(PR$,PO-1)+"    ",4)
200 REM spronginstructies apart behandelen ----------
210 IF B$="JP  " OR B$="JR  " THEN GOSUB 790:GOTO 460
220 IF B$<>"RST " THEN 240
230 AS=PC:PC=VAL(MID$(PR$,5,4)):GOTO 310
240 IF B$<>"DJNZ" THEN 280
250 B=PEEK(S1+15):B=B-1-256*(B=0):POKE S1+15,B:IF B<>0
 THEN GOSUB 870:GOTO 460 ELSE 460
260 IF B<>0 THEN GOSUB 870
270 GOTO 460
280 IF B$<>"CALL" THEN 370
290 AS=PC:GOSUB 790
300 REM terugsprongadres op stack leggen ----------
310 SP=PEEK(S3)+256*PEEK(S3+1)
320 SP=SP-1:POKE SP,INT(AS/256)-256*(AS<0)
330 SP=SP-1:POKE SP,AS-256*INT(AS/256)
340 POKE S3,SP-INT(SP/256)*256
```

```
 350 POKE S3+1,INT(SP/256)-256*(SP<0)
 360 GOTO 460
 370 IF LEFT$(B$,3)<>"RET" THEN 470
 380 GOSUB 720
 390 IF FL=0 THEN 460
 400 REM terugsprongadres van stack halen ----------
 410 SP=PEEK(S3)+256*PEEK(S3+1)
 420 PC=PEEK(SP+1)*256+PEEK(SP)
 430 SP=SP+2
 440 POKE S3,SP-INT(SP/256)*256
 450 POKE S3+1,INT(SP/256)-256*(SP<0)
 451 GOTO 460
 455 IF PEEK(&HFCAB)=255 THEN X=USR8(1):
REM echte simulatie aleen als CAP aan is ----------
 460 REM uitvoerregister ----------
 470 IF E$="j" OR E$="J" THEN POKE &HFEE4,&HC3
 480 PRINT"SZ H PNC A  B C  D E  H L   IX   IY "
 490 FOR Q=1 TO 0 STEP -1
 500 PRINTRIGHT$("0000000"+BIN$(PEEK(S1+Q*12)),8);
 510 PRINT" ";RIGHT$("0"+HEX$(PEEK(S1+1+Q*12)),2);
 520 FOR K=2 TO 6 STEP 2
 530 PO=K+Q*12:GOSUB 890:NEXT
 540 IF Q=1 THEN PO=8:GOSUB 890:PO=10:GOSUB 890
 ELSE PRINT" SP :";:PO=20:GOSUB 890
 550 PRINT:NEXT Q
 560 GOTO 80
 570 REM register veranderen met SELECT ----------
 580 LOCATE 0,CSRLIN-3
 590 LINE INPUT Z$
 600 Q=12:GOSUB 920
 610 Q=0:PO=28
 620 GOSUB 970
 630 PO=33:GOSUB 970
 640 LINE INPUT Z$
 650 Q=0:GOSUB 920
 660 Q=10:PO=33:GOSUB 970
 670 LINE INPUT Z$
 680 PC=VAL("&H"+LEFT$(Z$,4))
 690 LOCATE 0,CSRLIN-1
 700 GOTO 80
 710 REM subroutines ----------
 720 REM SUB conditietest ----------
 730 A$=MID$(PR$,PO+1,2)
 740 FOR I=0 TO 7:IF A$<>CO$(I) THEN NEXT
 750 IF I=8 THEN FL=1:RETURN
 760 BI=((I\2)*2-(I>3)-((I=4)OR(I=5)))
 770 IF(PEEK(S1+12)AND2^BI)/2^BI=-(IMOD2=1) THEN FL=-1 ELSE FL=0
 780 RETURN
```

```
 790 REM bij sprongen SUB PC zetten ----------
 800 GOSUB 720
 810 IF FL THEN  870
 820 IF FL=0 THEN RETURN
 830 A$=MID$(PR$,PO+2,2)
 840 IF A$="HL" THEN PC=PEEK(S1+18)+256*PEEK(S1+19):RETURN
 850 IF A$="IX" THEN PC=PEEK(S1+8)+256*PEEK(S1+9):RETURN
 860 IF A$="IY" THEN PC=PEEK(S1+10)+256*PEEK(S1+11):RETURN
 870 PC=VAL(RIGHT$(PR$,6)):RETURN
 880 REM SUB uitvoer 16-bits register ----------
 890 PRINT" ";RIGHT$("000"+HEX$(256*PEEK(S1+PO+1)
+PEEK(S1+PO)),4);
 900 RETURN
 910 REM SUB veranderde reg. naar machinetaalprog. ----------
 920 POKE S1+1+Q,VAL("&H"+MID$(Z$,10,2))
 930 POKE S1+Q,VAL("&B"+MID$(Z$,1,8))
 940 FOR PO=13 TO 23 STEP 5:GOSUB 970:NEXT
 950 RETURN
 960 REM SUB na verandering registerwaarden
 uit invoer bepalen ----------
 970 WE=VAL("&H"+MID$(Z$,PO,4))
 980 POKE S1+Q+(PO\5)*2-1,INT(WE/256)-256*(WE<0)
 990 POKE S1+Q+(PO\5)*2-2,WE-INT(WE/256)*256
1000 RETURN
1010 REM disassembler ----------
1020 AD=PC
1030 PRINTRIGHT$("000"+HEX$(AD),4);" ";
1040 IL=0
1050 GOSUB 1830
1060 GOSUB 1170
1070 IF IL THEN 1480
1080 IF LEFT$(PR$,3)="RST" AND (W=&HCF) THEN PR$=PR$+" /DB:n"
1090 IF INSTR(PR$,"n")<>0 THEN 1590
1100 IF INSTR(PR$,"e")<>0 THEN 1710
1110 PO=INSTR(PR$," ")
1120 IF PR$="" THEN PR$="???"
1130 IF PO=0 THEN PRINTTAB(15);PR$;:GOTO 1150
1140 PRINTTAB(15);LEFT$(PR$,PO-1);TAB(20);
RIGHT$(PR$,LEN(PR$)-PO);
1150 PRINT SPACE$(36-POS(0))
1160 RETURN
1170 REM interpreteren ----------
1180 IF (W=&HDD OR W=&HFD) AND NOT IL THEN 1370
1190 IF W=&HED THEN 1340
1200 IF W=&HCB THEN 1280
1210 GOSUB 1420
1220 ON C1 GOTO 1240,1260,1230
1230 PR$=BF$(W):RETURN
```

```
1240 IF W=&H76 THEN PR$="HALT":RETURN
1250 PR$="LD "+RT$(C2)+","+RG$:RETURN
1260 IF C2=0 OR C2=1 OR C2=3 THEN A$=" A," ELSE A$=" "
1270 PR$=AL$(C2)+A$+RG$:RETURN
1280 REM instructies beginnend met &HCB ----------
1290 GOSUB 1830
1300 IF IL THEN DI=W:GOSUB 1830
1310 GOSUB 1420
1320 IF C1=0 THEN PR$=RS$(C2)+" "+RG$
 ELSE PR$=BI$(C1)+STR$(C2)+","+RG$
1330 RETURN
1340 REM instructies beginnend met &HED ----------
1350 GOSUB 1830
1360 IF W<&H40 OR W>&HBF THEN PR$="???":RETURN ELSE GOTO 1230
1370 REM geindiceerde instructies ----------
1380 IL=-1
1390 IF W=&HDD THEN I$="IX" ELSE I$="IY"
1400 GOSUB 1830
1410 GOTO 1170
1420 REM code ontleden ----------
1430 C1=(W AND &B11000000)/64
1440 C2=(W AND &B111000)/8
1450 C3=W AND &B111
1460 RG$=RT$(C3)
1470 RETURN
1480 REM geindiceerd ----------
1490 PO=INSTR(PR$,"HL")
1500 IF PO=0 THEN PR$="???":GOTO 1110
1510 IF INSTR(PR$,"(HL)")<>0 THEN 1550
1520 IF PR$="EX DE,HL" THEN PR$="???":GOTO 1110
1530 IF PR$="ADD HL,HL" THEN PR$="ADD "+I$+","+I$:GOTO 1110
1540 PR$=LEFT$(PR$,PO-1)+I$+RIGHT$(PR$,LEN(PR$)-PO-1):GOTO 1080
1550 IF LEFT$(PR$,2)="JP" THEN 1540
1560 IF PC-ADR<3 THEN GOSUB 1830:DI=W
1570 IF DI>127 THEN DI$=STR$(DI-256) ELSE
 DI$="+"+RIGHT$(STR$(DI),LEN(STR$(DI))-1)
1580 I$=I$+DI$:GOTO 1540
1590 REM n vervangen ----------
1600 PO=INSTR(PR$,"nn")
1610 IF PO<>0 THEN 1660
1620 PO=INSTR(PR$,"n")
1630 GOSUB 1830
1640 PR$=LEFT$(PR$,PO-1)+"&H"+RIGHT$("00"+HEX$(W),2)
+RIGHT$(PR$,LEN(PR$)-PO)
1650 GOTO 1110
1660 GOSUB 1830:LB=W
1670 GOSUB 1830
1680 WE=W*256+LB
```

```
1690 PR$=LEFT$(PR$,PO-1)+"&H"+RIGHT$("0000"+HEX$(WE),4)
+RIGHT$(PR$,LEN(PR$)-PO-1)
1700 GOTO 1110
1710 REM e vervangen ----------
1720 PO=INSTR(PR$,"e")
1730 GOSUB 1830
1740 IF W>127 THEN W=W-256:REM twee-complement ----------
1750 W=W+2
1760 A$="$"+STR$(W)+" >"+"&H"+RIGHT$("0000"+HEX$(PC+W-2),4)
1770 PR$=LEFT$(PR$,PO-1)+A$+RIGHT$(PR$,LEN(PR$)-PO)
1780 GOTO 1110
1790 REM hexadecimaal omzetten in decimaal ----------
1800 IF A$="" THEN A=0:RETURN
1810 A=VAL("&H"+A$)
1820 RETURN
1830 REM byte lezen ----------
1840 W=PEEK(PC)
1850 POKE S4+QQ,W:QQ=QQ+1
1860 PC=PC+1
1870 PRINTRIGHT$("00"+HEX$(W),2);
1880 RETURN
1890 REM variabelen initialiseren ----------
1900 DIM RT$(7),RS$(8),BI$(3),AL$(7),BF$(255)
1910 FOR I=0 TO 7: READ RT$(I):NEXT
1920 FOR I=0 TO 7: READ RS$(I):NEXT
1930 FOR I=1 TO 3: READ BI$(I):NEXT
1940 FOR I=0 TO 7: READ AL$(I):NEXT
1950 FOR I=0 TO &H7F: READ BF$(I):NEXT
1960 FOR I=&H80 TO &H9F:BF$(I)="":NEXT
1970 FOR I=&HA0 TO &HFF:READ BF$(I):NEXT
1980 GOTO 2300
1990 REM instructietabellen ----------
2000 DATA B,C,D,E,H,L,(HL),A
2010 DATA RLC,RRC,RL,RR,SLA,SRA,???,SRL
2020 DATA BIT,RES,SET
2030 DATA ADD,ADC,SUB,SBC,AND,XOR,OR,CP
2040 DATA NOP,"LD BC,nn","LD (BC),A",INC BC,INC B,
DEC B,"LD B,n",RLCA
2050 DATA "EX AF,AF'","ADD HL,BC","LD A,BC",DEC BC,
INC C,DEC C,"LD C,n",RRCA
2060 DATA DJNZ e,"LD DE,nn","LD (DE),A",INC DE,INC D,
DEC D,"LD D,n",RLA
2070 DATA JR e,"ADD HL,DE","LD A,(DE)",DEC DE,INC E,
DEC E,"LD E,n",RRA
2080 DATA "JR NZ,e","LD HL,nn","LD (nn),HL",INC HL,
INC H,DEC H,"LD H,n",DAA
2090 DATA "JR Z,e","ADD HL,HL","LD HL,(nn)",DEC HL,
INC L,DEC L,"LD L,n",CPL
```

```
2100 DATA "JR NC,e","LD SP,nn","LD (nn),A",INC SP,
INC (HL),DEC (HL),"LD (HL),n",SCF
2110 DATA "JR C,e","ADD HL,SP","LD A,(nn)",DEC SP,
INC A,DEC A,"LD A,N",CCF
2120 DATA "IN B,(C)","OUT (C),B","SBC HL,BC",
"LD (nn),BC",NEG,RETN,IM 0,"LD I,A"
2130 DATA "IN C,(C)","OUT (C),C","ADC HL,BC",
"LD BC,(nn)",,RETI,,"LD R,A"
2140 DATA "IN D,(C)","OUT (C),D","SBC HL,DE",
"LD (nn),DE",,,IM 1,"LD A,I"
2150 DATA "IN E,(C)","OUT (C),E","ADC HL,DE",
"LD DE,(nn)",,,IM 2,"LD A,R"
2160 DATA "IN H,(C)","OUT (C),H","SBC HL,HL",
"LD (nn),HL",,,,RRD
2170 DATA "IN L,(C)","OUT (C),L","ADC HL,HL",
"LD HL,(nn)",,,,RLD
2180 DATA ,,"SBC HL,SP","LD (nn),SP",,,,
2190 DATA "IN A,(C)","OUT (C),A","ADC HL,SP",
"LD SP,(nn)",,,,
2200 DATA LDI,CPI,INI,OUTI,,,,,LDD,CPD,IND,OUTD,,,,
2210 DATA LDIR,CPIR,INIR,OTIR,,,,,LDDR,CPDR,INDR,OTDR,,,,
2220 DATA RET NZ,POP BC,"JP NZ,nn",JP nn,"CALL NZ,nn",
PUSH BC,"ADD A,n",RST &H00
2230 DATA RET Z,RET,"JP Z,nn",->,"CALL Z,nn",CALL nn,
"ADC A,n",RST &H08
2240 DATA RET NC,POP DE,"JP NC,nn","OUT (n),A","CALL NC,nn",
PUSH DE,"SUB n",RST &H10
2250 DATA RET C,EXX,"JP C,nn","IN A,(n)","CALL C,nn",->,
"SBC A,n",RST &H18
2260 DATA RET PO,POP HL,"JP PO,nn","EX (SP),HL","CALL PO,nn",
PUSH HL,"AND n",RST &H20
2270 DATA RET PE,JP (HL),"JP PE,nn","EX DE,HL","CALL PE,nn",
->,"XOR n",RST &H28
2280 DATA RET P,POP AF,"JP P,nn",DI,"CALL P,nn",PUSH AF,
"OR n",RST &H30
2290 DATA RET M,"LD SP,HL","JP M,nn",EI,"CALL M,nn",->,
"CP n",RST &H38
2300 ON STOP GOSUB 2520
2310 STOP ON
2320 ON ERROR GOTO 2520
2330 POKE &HFEE5,&H70:POKE &HFEE6,&HF3
2340 FOR I=&HF370 TO &HF37B:READ A$:POKE I,VAL("&H"+A$):NEXT
2350 DATA F5,3A,61,F6,32,15,F4,F1
2360 DATA CD,63,1B,C9
2370 FOR I=0 TO 7:READ CO$(I):NEXT
2380 DATA NC,"C,",PO,PE,NZ,"Z,","P,","M,"
2390 FOR I=&HF200 TO &HF236:READ A$:POKE I,VAL("&H"+A$):NEXT
2400 DATA ED,73,37,F2,31,39,F2,F1
```

```
2410 DATA C1,D1,E1,D9,08,DD,E1,FD
2420 DATA E1,F1,C1,D1,E1,ED,7B,4D
2430 DATA F2,00,00,00,00,ED,73,4D
2440 DATA F2,31,4D,F2,E5,D5,C5,F5
2450 DATA FD,E5,DD,E5,08,D9,E5,D5
2460 DATA C5,F5,ED,7B,37,F2,C9
2470 DEFUSR8=&HF200
2480 S1=&HF239:S2=&HF24C:S3=&HF24D:S4=&HF219
2490 FOR I=S1 TO S2:POKE I,0:NEXT
2500 POKE S3,&H90:POKE S3+1,&HF2
2510 GOTO 50
2520 POKE &HFEE4,&HC9:ON ERROR GOTO 0:LOCATE 0,24,0:END
```

5.3.3 Programmabeschrijving

regel 10-60

Initialisering.

regel 70-150

Hoofdlus.

regel 80

Printer inschakelen als E$="j" of "J".

regel 90

Bereik instructie-overdracht wissen (NOP).

regel 100

Instructie disassembleren, uitvoeren en de code in het machinetaalprogramma POKEn.

regel 110

Printer uitschakelen.

regel 120-160

Invoerlus met interpretatie.

regel 170

Instructie niet uitvoeren als CAP uit is.

regel 180-190

Instructiewoord naar B$.

regel 200-450

Spronginstructies simuleren vanuit BASIC.

regel 210

JP/JR

regel 230

RST

regel 250

DJNZ verhogen met B.

regel 290

CALL

regel 300-360

Aanpassing van stack, SP en PC bij subroutinesprong (CALL of RST).

regel 380-390

RET

regel 400-450

Aanpassing van stack, SP en PC bij sprong terug (RET).

regel 455

Echte simulatie door een machinetaalprogramma (op voorwaarde dat CAP aan is en er geen spronginstructie wordt gesimuleerd).

regel 460-560

Geformateerde uitvoer van de inhoud van de registers.

regel 570-700

Invoerroutine, als registers worden veranderd.

regel 710-1000

Subroutines.

regel 720-780

SUB(routine) die de volgende waarden voor FL oplevert:

1	als er helemaal geen voorwaarde is
0	als niet is voldaan aan de voorwaarde
-1	als wel is voldaan aan de voorwaarde

regel 760

Bepaalt het bitnummer van de actuele flag in het flagregister.

regel 790-860

PC wordt gezet afhankelijk van FL. Indirecte sprongen worden in regel 830-860 behandeld.

regel 870

PC wijst op het aangegeven adres.

regel 880-890

Voert een 16-bits register uit. De plaats van de registers volgt uit de volgorde van de instructies PUSH of POP in het machinetaalprogramma.

regel 910-950

De standaardregisters worden na aanpassing opgeslagen in de betreffende geheugenplaatsen.

regel 960-1000

Met deze routine kunt u na een verandering een willekeurig register opnieuw zetten.

regel 1010-1880

De disassembler (zie voor een beschrijving aldaar). Ten opzichte van de oorsponkelijke versie is het volgende gewijzigd:

- het uitvoerformaat is enigszins anders (regel 1180)
- gelezen codes worden direct in de betreffende plaats in het machinetaalprogramma gePOKEt (regel 1850)
- het uitvoerformaat van de codes (regel 1870) is veranderd

regel 1890-2340

Initialisering. Tot aan regel 2340 hetzelfde als bij disassembler.

regel 2350-2540

De speciaal voor de simulator benodigde variabelen en het machinetaalprogramma van de simulator inlezen.

Variabelenlijst

A	resultaat van subroutine voor hex --> dec. Waarde van A$ geïnterpreteerd als hex-getal
A$	overdracht van subroutines
AD	adres van de eerste code van de actuele instructie
AS	sprongadres
B	waarde in register B
B$	instructiewoord
BI	bitnummer
C1	bit 7 en 6
C2	bit 5-3
C3	bit 2-0
DI	afstand (distance) bij geïndiceerde instructies
DI$	afstandsstring voor uitvoer
E$	invoerstring (j/n)
FL	flag voor test van een conditie
I$	actuele indexregister
IL	gezet bij geïndiceerde adressering, anders gereset
K	teller
LB	buffer met lowbyte: bij 2-bytes getallen
PC	programmateller: wijst op adressen van de actuele positie
PO	positie van n, nn, e, H... in PR$
PR$	print $, bevat vertaalde assemblerinstructie
Q	teller
QQ	teller
RG$	register: resultaat van subroutine die code splitst; RG$ bevat het register dat bij C3 hoort
S1	registeroverdracht van beginadres
S2	einde registeroverdracht van beginadres
S3	overdrachtsadres stack
S4	overdrachtsadres instructiecode
SP	stackadres
W	gelezen code
WA	waarde van een 2-bytes getal (nn)
Z$	invoerregel

RT$(7)	registers	
RS$(7)	roteer- en schuifinstructies	
BI$(3)	bitmanipulatie-instructies	
AL$(7)	logische en rekeninstructies	
BF$(255)	0 -&H3F:	instructies die met &HED beginnen en als byte 1 het nummer van de veldvariabelenteller bevatten
	&H40-&HBF:	instructies die met &HED beginnen en als byte 2 dat nummer bevatten
	&HBF-&HFF:	instructies die niet met &HED beginnen en als byte 1 dat nummer bevatten
CO$(7)	voorwaarde	

5.4 De instructie USR

In tegenstellimg tot wat elders gebruikelijk is, heeft bij de instructie USR in MSX-BASIC een directe overdracht van parameters plaats. Om die mogelijkheid goed te kunnen gebruiken, moet u de interne weergave van getallen en alfanumerieke gegevens (strings) goed leren kennen. Er zijn vier manieren om computergegevens weer te geven.

1	integer	INT	gehele getallen
2	single precision	SNG	enkele nauwkeurigheid
3	double precision	DBL	dubbele nauwkeurigheid
4	strings	STR	alfanumeriek

Integere getallen

Eerst een korte samenvatting van de al besproken weergave van integere getallen. Een integer getal wordt gesplitst in een low- en een highbyte. Bit 7 van de highbyte geeft het voorteken aan: 0 betekent positief en 1 negatief. De negatieve getallen worden weergegeven als twee-complement: nadat het complement is berekend, tellen we er 1 bij op (zie paragraaf 4.6). Integere constanten of variabelen kunnen waarden van -32768 tot +32767 hebben.

decimaal	binair	hex
-32768	1000 0000 0000 0000	80 00
-32767	1000 0000 0000 0001	80 01
-32766	1000 0000 0000 0010	80 02
-32765	1000 0000 0000 0011	80 03
	.....	
-2	1111 1111 1111 1110	FF FE
-1	1111 1111 1111 1111	FF FF
0	0000 0000 0000 0000	00 00
1	0000 0000 0000 0001	00 01
2	0000 0000 0000 0010	00 10
	.....	
32766	0111 1111 1111 1110	7F FE
32767	0111 1111 1111 1111	7F FF

Aan elk type gegevens is een kenmerk toegevoegd om de types intern van elkaar te kunnen onderscheiden. Dit kenmerk is het aantal bytes, nodig om het betreffende type op te bergen. Voor een integer getal zijn twee bytes nodig, dus is het kenmerk 2. De systeemroutine van de BASIC-interpreter die de instructie USR uitvoert, laadt het kenmerk in de accu. Bovendien staat het kenmerk van de actuele variabele altijd op adres &HF663. Zo kunt u controleren of de overgedragen parameter van het goede type is. De beide bytes gaan naar de adressen &HF7F8 en &HF7F9. Nu we weten hoe een waarde in het machinetaalprogramma komt te staan, kunnen we een paar eenvoudige instructie-uitbreidingen tot stand brengen.

DEEK

In assembler is de functie FNDK gedefinieerd. Deze leest twee opeenvolgende geheugenplaatsen als een 16-bits waarde. Sommige BASIC-dialecten hebben voor deze taak de instructie DEEK. Met behulp van USR voegen we DEEK toe aan MSX-BASIC. Het adres van de lowbyte van de twee te lezen geheugenplaatsen moet worden overgedragen. Eerst controleert het programma of de overgedragen parameter van het goede type is. Zoniet, dan volgt een sprong naar de uitvoer van "Type mismatch error". Het nummer van de betreffende fout wordt in register E geladen en er volgt een sprong naar &H406F, het adres van de foutmeldingsroutine.

```
CP 2                    ; type 2?
JR Z,OK                 ; ja, dan OK
LD E,13                 ; nee, dan
```

```
JP &H406F                    ; Type mismatch error
OK ...
```

Iedere foutmelding heeft een eigen sprongadres, dat ervoor zorgt dat register E wordt geladen. Voor de boodschap "Type mismatch error" is dat adres &H406D. Het programma wordt daarmee gereduceerd tot:

```
CP 2                         ; type 2?
JP NZ,&H406D                 ; nee, dan "Type mismatch error"
```

Het type op deze manier controleren heeft een nadeel. Bij invoer van een getal van de juiste grootte maar van een verkeerd type (1000 bijvoorbeeld kan als INT-, SNG- of DBL-type worden opgeslagen) geeft het programma "Type mismatch error". Zo'n getal kan altijd worden omgezet in een integer getal en zo worden opgeslagen. In BASIC gebeurt dat met de functie CINT. Als we in een machinetaalprogramma aan het begin CINT aanroepen, gaat die omzetting verder automatisch. De CINT-routine gaat ook na of een getal niet te groot is. In dat geval volgt de foutmelding "Overflow". Fouten van het hiervoor behandelde type treden dan alleen nog op bij stringinvoer. Als u andere typen variabelen nodig hebt, gebruik dan de routines CSNG (&H2FB2) en CDBL (&H303A). In de tabellen achterin het boek staan onder andere de hier behandelde systeemroutines vermeld. Over naar de uitvoering van de instructie DEEK.

```
10 ' CALL &H2F8A             ; CINT zet getal om in INT
20 ' LD,(&HF7F8)             ; overgedragen par.waarde (adr.)
30 ' LD E,(HL)               ; lowbyte
40 ' INC HL
50 ' LD D,(HL)               ; highbyte
```

In regel 20 leest de computer de parameterwaarde die moet worden overgedragen (het adres dus) uit de overdrachtsadressen &HF7F8 en &HF7F9. In de regels 30-50 wordt die 2-bytes waarde in register DE geladen. Vervolgens moet de verkregen waarde weer aan het BASIC-programma worden overgedragen. De over te dragen waarde moet worden geladen in de adressen &HF7F8/9. Bovendien moet de accu het typenummer bevatten en moet HL worden geladen met &HF7F6.

```
60 ' LD (&HF7F8),DE          ; resultaat van DEEK
70 ' LD HL,&HF7F6
80 ' RET
```

Assembleer dit programma en pas de nieuwe functie toe.

```
?USR1(2)
```

levert 370 op, hetzelfde resultaat als

```
?PEEK(2)+256*PEEK(3)
```

Dat wil zeggen dat vanaf adres 2 de 2-bytes waarde 370 staat, oftewel &H207. Tot zover de overdracht van integere getallen. De beide andere representaties van getallen zijn fundamenteel anders.

Getallen met drijvende komma

Het bedrijfssysteem van de MSX gebruikt het BCD-formaat om getallen met een drijvende komma op te slaan. Dat is heel ongewoon voor computers in deze prijsklasse. In het binary coded decimal-systeem wordt elk cijfer van een decimaal getal apart opgeslagen. Zo ligt precies vast hoeveel posities kunnen worden verwerkt. Bovendien is bij de MSX het bereik van de waarden (10^-64 tot 10^62) veel groter dan bij de normale methode (10^-32 tot 10^31). Daar staat tegenover dat BCD-getallen zich niet zo snel laten verwerken. Er zijn daarom twee graden van nauwkeurigheid: SNG (6 posities) en DBL (14 posities). De representatie met enkelvoudige nauwkeurigheid (SNG) voldoet bijna altijd aan de gestelde eisen. Om te verduidelijken hoe een getal in BCD-formaat wordt opgeslagen, bespreken we de exponentiële schrijfwijze, zoals die bijvoorbeeld wordt toegepast in zakrekenmachientjes. We stellen eerst vast hoe vaak de basis van het decimale systeem (10) als factor voorkomt in het getal. Vervolgens splitsen we het getal in twee gedeelten:

- de mantisse	alle cijfers, met de komma achter het eerste cijfer
- de macht van tien	de exponent bepaalt het aantal plaatsen dat de komma moet verschuiven vergeleken met de mantisse

Dat gaat als volgt:

```
  27 =   2.7*10^1

3956 = 3.956*10^4
```

We spreken over komma's maar schrijven punten, zoals gebruikelijk is in de VS. Verder geldt de afspraak dat met negatieve exponenten een verplaatsing van de komma naar links wordt aangegeven. Zo kunnen we alle getallen weergeven.

```
    0.21 =  2.1*10^-1

  0.0051 =  5.1*10^-3

      -9 = -9.0*10^0
```

Het grote voordeel van de exponentiële schrijfwijze is dat zeer grote en zeer kleine getallen relatief weinig ruimte innemen.

```
0.00000000000000735 = 7.35*10^-15

   6390000000000000 = 6.39*10^15
```

Bij berekeningen met deze getallen gelden de gebruikelijke regels van exponentrekenen.

Optellen en aftrekken

Deze bewerkingen zijn alleen mogelijk bij getallen met een gelijke exponent. Als ze verschillend zijn, herschrijft men het getal met de kleinste exponent naar een vorm met een exponent gelijk aan de grootste. Daarna mogen de mantissen worden opgeteld of afgetrokken.

```
      57 + 0.31
5.7*10^1 + 3.1*10-1
5.7*10^1 + 0.031*10^1
   5.731 * 10^1
       57.31
```

Vermenigvuldigen en delen

Bij vermenigvuldigen of delen in de wetenschappelijke notatie vermenigvuldigt of deelt u eerst de mantissen waarna u de exponenten van de machten van 10 respectievelijk optelt of aftrekt.

```
     0.13 * 20
1.3*10^-1 * 2.0*10^1
  1.3*2.0 * 10^(-1+1)
      2.6 * 10^0
      2.6
```

```
        25 / 0.05
  2.5*10^1 / 5.0*10^-2
   2.5/5.0 * 10^(-1-(-2))
       0.5 * 10^3
       5.0 * 10^2
       500
```

Om een microprocessor op deze manier te laten werken, moeten de getallen in het wetenschappelijke formaat in het geheugen worden geplaatst. In het BCD-systeem wordt elk cijfer van de mantisse apart opgeslagen. De waarde van een teken in het decimale systeem is maximaal 9 (&B1001), zodat er vier bits nodig zijn voor een cijfer. Een byte kan dus twee cijfers bevatten: het eerste is de highnibble (bit 4-7) en het tweede de lownibble (bit 0-3).

```
decimaal      27
highnibble    2 =&B10
lownibble     7 =&B111
BCD-formaat   &B0010 0111=&H27 (=39)
```

Omdat in het hexadecimale systeem vier bits een hexadecimaal cijfer vertegenwoordigen, is de waarde van het als hexadecimaal gelezen decimale getal altijd precies de waarde in de BCD-code. De BCD-waarde van 63 is &H63 = &B0110 0011 (=99). Het aantal cijfers dat we in het geheugen kunnen plaatsen, is begrensd door het aantal bytes dat we ervoor reserveren. Dat zijn er drie bij rekenoperaties met enkelvoudige nauwkeurigheid (SNG) zodat er per mantisse zes cijfers beschikbaar zijn (twee per byte). In BCD-formaat staat het getal 123794 als een reeks van drie bytes in het geheugen: &H12, &H37, &H94. Als u kiest voor dubbele nauwkeurigheid gaat dat met 7 bytes oftewel 14 cijfers. Voor de codering van de exponent is een andere keus gemaakt. Hier is de waarde van een byte gelijk aan de exponent. Verder moeten we nog bepalen wat de tekens van de mantisse en de exponent zijn. Het voorteken van het getal staat zoals altijd in bit 7:

```
0     positief
1     negatief
```

De overige 7 bits geven de waarde van de exponent aan, waarbij aan de feitelijke waarde altijd &H41 is toegevoegd.

waarde bit 6-0	waarde exponent
&H7F	&H3E = 62
&H7E	&H3D = 61
..	..
..	..
&H42	&H01 = 1
&H41	&H00 = 0
&H40	-&H01 = -1
..	..
..	..
&H02	-&H3F = -63
&H01	-&H40 = -64
&H00	betekent: getal = 0

De exponent kan dus lopen van -64 tot +62; ruim voldoende voor alle berekeningen. Bij afspraak krijgt een getal met de waarde 0 ook een exponent gelijk aan 0. We passen deze schrijfwijze toe op een BASIC-programma. De functie VARPTR levert het adres waar de variabelen staan als ze gecodeerd zijn op de besproken manier. Voor de interpretatie van de variabelen volgt direct na het programma een bereik waar van iedere gebruikte variabele naam en waarde zijn opgeslagen. VARPTR wijst naar het adres met de eerste byte van het gecodeerde getal, die volgens de afspraken de betekenis 'exponent' heeft. Daarop volgen, afhankelijk van het type, de 3 (SNG) of 7 (DBL) bytes die de cijfers van de mantisse weergeven. Neem verschillende waarden voor X om te bekijken welke invloed enkele (!) en dubbele (#) precisie op de variabelen hebben in de volgende programma's.

```
10 X!=43546!
20 AD=VARPTR(X!)
30 PRINT HEX$(PEEK(AD))
40 FOR I=AD+1 TO AD+3
50 PRINT RIGHT$("0"+HEX$(PEEK(I)),2);
60 NEXT
```

```
10 X#=.012342349#
20 AD=VARPTR(X#)
30 PRINT HEX$(PEEK(AD))
40 FOR I=AD+1 TO AD+6
50 PRINT RIGHT$("0"+HEX$(PEEK(I)),2);
60 NEXT
```

Over het algemeen krijgen getallen bij hun interne verwerking een plaats in een speciale buffer in het RAM. Dit bereik heet FAC

(floating point accu). In MSX-BASIC loopt dat bereik van adres &HF7F6 tot &HF7FD. Het systeem heeft minstens twee van deze buffers om bijvoorbeeld de termen van een optelling op te slaan. Ook het resultaat komt in de FAC te staan. Een ander belangrijk adres is &HF663, waar het typenummer van de actuele variabele staat. De systeemroutine USR laadt dit nummer in de accu. Bovendien komt het startadres (&HF7F6) van de FAC in register HL. Als we de overgedragen parameter willen gebruiken, moeten we de inhoud van de FAC benutten. Het geheugenformaat van de FAC ziet er zo uit:

type	&HF7F6	&HF7F7	&HF7F8	&HF7F9	&H...	&HF7FD
2	------	------	highb.	lowb.	-----	------
4	exponent	m a n	t i	s s e	-----	------
8	exponent	m a	n t	i	s	s e

Als aan het einde van het programma een waarde als resultaat moet worden teruggegeven, moeten we register en geheugen weer op de boven besproken wijze laden:

accu	code van het type
register HL	beginadres FAC = &HF7F6
FAC	gecodeerde waarde

BCD-instructies voor de Z80

Om een idee te krijgen van het rekenwerk in de computer is het nuttig het volgende programma eens te bekijken. In regel 80 staat de belangrijke instructie DAA (decimal adjust accu). Kijk wat er gebeurt als u deze regel weglaat, bijvoorbeeld bij overdracht van het getal 123456789, dus met ?USR1(123456789).

```
F000           10                     ; maal 2
F000 CD3A30    20       CALL &H303A   ; zet om in
                                      ; DBL-formaat
F003 21FDF7    30       LD   HL,&HF7FD ; einde FAC
F006 0607      40       LD   B,7      ; aantal bytes
F008 B7        50       OR   A        ; carry wissen
F009 7E        60  NEXT LD   A,(HL)   ; byte-waarde
F00A 8F        70       ADC  A,A      ; verdubbelen
F00B 27        80       DAA           ; BCD-correctie
F00C 77        90       LD   (HL),A   ; weer opslaan
F00D 2B       100       DEC  HL       ; adres volgende
                                      ; byte
F00E 10F9     110       DJNZ NEXT     ; 7 maal herhalen
F010 3E08     120       LD   A,8      ; typenummer
```

```
F012 21F6F7        LD   HL,&HF7F6 ; adres van FAC
F015 C9            RET
```

```
programma: maal 2
start: &HF000
einde: &HF015
lengte: &H16
fouten: 0
variabelentabel:
NEXT      F009
```

De instructie DAA zorgt ervoor dat na operaties in BCD-formaat (ADD, SUB, ADC en SBC) eventuele fouten gecorrigeerd worden.

Voorbeeld

```
        7+7 = 14 (decimaal)
&H07+&H07 = &H0E<>&H14
```

Hier stuiten we op een verschil tussen de rekenregels van het hexadecimale stelsel en die van het BCD-systeem. De instructie DAA verandert in dit geval het foute resultaat &H0E in het juiste &H14. Bovendien krijgen de flags de waarde die overeenstemt met het werkelijke resultaat. U kunt DAA gebruiken na zowel optel- als aftrekinstructies. Intern worden deze instructies uitgevoerd met behulp van de aftrekflag (N), die 1 is bij een aftrekking en 0 bij een optelling. DAA regelt de toestand van de flags automatisch, evenals de tweede interne flag (H) van de halve overdracht. H is gezet als een overdracht plaatsvindt van bit 3 naar bit 4, of anders gezegd van het ene BCD-cijfer naar het andere. Als u bovenstaand programma met de simulator doorloopt om de werking van de instructie DAA na te gaan, moet u de instructie LD (HL),A niet daadwerkelijk uitvoeren (CAP uitzetten). We willen ook de aandacht vestigen op de BCD-rotatie-instructies RRD en RLD. Bij deze instructies gaat het om de rotatie van vier bits, tussen de twee cijfers (ieder 4 bits) op het adres waar het HL-register naar verwijst en 1 cijfer in de onderste helft (bit 0-30) van de accu.

Voorbeeld

```
A=&H23    (HL)=&H45 (byte op adres HL)

na RRD:

A=&H25    (HL)=&H34

na RLD:

A=&H24    (HL)=&H53
```

Strings

In een string worden alfanumerieke gegevens (letters, cijfers en tekens) opgeslagen. De code (0-127) die bij ieder teken hoort is voor de MSX praktisch gelijk aan de american standard code for information interchange (ASCII). In het geheugen neemt een teken precies 1 byte in beslag. Een string is een reeks opeenvolgende tekencodes. In het RAM staat een string op een serie opeenvolgende geheugenplaatsen. Om een string gelijk te stellen aan een bepaalde variabele hebben we twee gegevens nodig die samen de stringdescriptor vormen.

- het adres van de geheugenplaats met het eerste teken van de string
- de lengte van de string, dus het aantal bytes van de tekenreeks

Samen met de naam van de betreffende variabele staan deze gegevens in het bereik van de BASIC-variabelen. De tekenreeks zelf staat dus op een andere plek: in het programma of in een speciaal daarvoor gereserveerd stringbereik. De grenzen van dit gebied kunt u bepalen met de instructie CLEAR. In BASIC dient de instructie VARPTR om het adres van de stringdescriptor van een variabele te lezen. Een stringdescriptor bestaat uit 3 bytes:

- byte 1 — lengte van de string
- byte 2 + 3 — startadres van de string

De functie VARPTR voert het adres van de eerste byte van de stringdescriptor uit. Test het volgende programmaatje.

```
10 X$="TTTekenrEEEEks"
20 AD=VARPTR(X$)
30 LA=PEEK(AD)
40 ST=PEEK(AD+1)+256*PEEK(AD+2)
```

```
50 FOR I=ST TO ST+LA-1
60 PRINT CHR$(PEEK(I));
70 NEXT
```

Met betrekking tot de instructie USR vermelden we nog dat alfanumerieke strings typenummer 3 hebben omdat de stringdescriptor drie bytes beslaat. Zoals gezegd gaat dit naar de accu en adres &HF663. Het adres waar de stringdescriptor staat (het resultaat van VARPTR in BASIC, zie boven) wordt overgedragen aan register DE. De string weer overdragen aan BASIC gaat op precies dezelfde manier. Gewapend met deze informatie kunt u bijvoorbeeld de in sommige BASIC-dialecten voorkomende instructie UPPER uitvoeren. Deze instructie verandert alle kleine letters in een string in hoofdletters. Die omzetting zijn we al in de paragraaf over logische instructies tegengekomen (4.8). Schrijf met behulp van het programma in paragraaf 4.8 een programma dat de instructie UPPER uitvoert als een USR-instructie in MSX-BASIC.

```
 F000            10                      ; UPPER-instructie
 F000            20          ORG  &HF000
 F000            30  TYPERR  EQU  &H406D     ; type mismatch error
 F000 FE03       40          CP   3          ; stringvariabele?
 F002 C26D40     50          JP   NZ,TYPERR  ; nee dan Type
                                             ; mismatch error
 F005 1A         60          LD   A,(DE)     ; lengte v.d. strings
 F006 B7         70          OR   A          ; zet z-flag als a=0
 F007 C8         80          RET  Z          ; klaar als lengte=0
 F008 47         90          LD   B,A        ; lengte als lusteller
 F009 62        100          LD   H,D        ; descriptoradres
 F00A 6B        110          LD   L,E        ; naar HL
 F00B 23        120          INC  HL
 F00C 7E        130          LD   A,(HL)     ; adres
 F00D 23        140          INC  HL         ; van het
 F00E 66        150          LD   H,(HL)     ; stringbegin
 F00F 6F        160          LD   L,A        ; naar HL
 F010 7E        170  LUS     LD   A,(HL)     ; wanneer ASCII-code
 F011 FE7B      180          CP   123        ; >= aan  ASCII("z")+1
?F013 30FE      190          JR   NC,OK      ; dan niet verwisselen
 F015 FE61      200          CP   97         ; wanneer < ASCII("a:)
?F017 38FE      210          JR   C,OK       ; dan niet verwisselen
 F019 E6DF      220          AND  &B11011111; verwisseling
**** regel  190  : OK=&HF01B offset 6
**** regel  210  : OK=&HF01B offset 2
 F01B 77        230  OK      LD   (HL),A     ; ASCII weer opslaan
 F01C 23        240          INC  HL         ; adres v. volg. code
 F01D 10F1      250          DJNZ LUS        ; tot eind v. string
                                             ; herhalen
 F01F 3E03      260          LD   A,3        ; kengetal stringtype
```

```
 F021            270                          ; oude descr.ad. in DE
 F021 C9         280          RET
-----------------------------------------------------------------------
```

programma: UPPER
begin: &HF000   einde: &HF021
lengte: &H22 bytes
fout: 0
variabelen:
TYPERR 406D   LUS   F010
OK   F01B

SLOT, PEEK

Tot slot van dit hoofdstuk bespreken we manieren waarmee de mogelijkheden van de instructie USR zover worden uitgebreid dat meerdere parameters kunnen worden overgedragen. Daarvoor moeten we nog een paar fundamentele aspecten van de BASIC-interpreter bespreken. Een BASIC-programma wordt na invoer vertaald in een tussencode. Een instructie wordt bijvoorbeeld niet als zodanig opgeslagen, maar als verkorte code (1 byte lang en >128), het zogenaamde token. Als de interpreter een token herkent, springt hij naar de routine die hoort bij het token (de instructie). Eventueel worden daar de parameters na de instructie ingelezen en op juistheid getoetst. We nemen als voorbeeld de instructie POKE. Erachter staan twee parameters, door een komma gescheiden: het adres en de waarde. We nemen aan dat het token voor POKE (152) is herkend en dat het programma naar de POKE-routine springt (&H5423). Bij de interpretatie van BASIC-programma's wijst het register HL steeds op de plaats in het programma die wordt bewerkt (HL dient als BASIC program pointer). We moeten ervoor zorgen dat deze pointer niet weg raakt, omdat anders de interpretatie niet meer correct wordt uitgevoerd. Om bovengenoemde parameters te lezen, kunnen we ook speciaal daarvoor in de interpreter aanwezige routines gebruiken. Bij de instructie POKE wordt eerst de routine GETADR aangeroepen. Deze leest een integere parameter. Wordt deze routine aangeroepen, dan moet HL met de actuele BASIC-pointer worden geladen. De routine maakt bij de interpretatie geen onderscheid tussen een simpele uitdrukking als

```
1237
```

of een meer complexe uitdrukking als

```
ABS(INT(VAL(RIGHT$(STRING$(4,"0")+HEX$(x),4))))
```

Het resultaat, een 2-bytes waarde, gaat naar register DE. Na de routine wijst de BASIC-pointer HL in dit geval naar de volgende

komma. Het adres van de routine GETADR is &H542F. Vervolgens test het programma of hierna een komma volgt. Ook hiervoor is uiteraard een systeemroutine: RST &H08. Deze restart test op een willekeurig teken. Op de instructie RST &H08 volgt direct DB met de ASCII-code van het teken. Voor de POKE-instructie wordt dat dus

```
    ..
    ..
RST &H08                    ; test op ","
DB  &H2C                    ; code van ","
    ..
    ..
```

Als het betreffende teken niet wordt gevonden, volgt de foutmelding 'Syntax error', en anders een sprong terug. Dan wordt de routine GETBYT aangeroepen om de 8-bits waarde te lezen en door te geven aan de accu. Na afloop van de routine wijst de BASIC-pointer HL op het einde van de instructie. Met deze routines kunt u de instructie USR uitbreiden om meerdere parameters over te dragen. In het volgende programma maken we er gebruik van om een aangepaste versie van de instructie PEEK te produceren die behalve het adres ook het slotnummer aangeeft. Het formaat van deze instructie is

```
X=USR1(adres),slot
```

bijvoorbeeld:

```
X=USR1(&HF000),2
```

2 is de defaultwaarde van het slotnummer

Met deze instructie kunnen uitbreidingsmodules en ook ingebouwde, elkaar overlappende modules worden gelezen. De routine SLOT RD (vanaf &H000C) zorgt voor de feitelijke uitvoering. Als het slotnummer wordt overgedragen aan de accu en het adres aan HL, draagt deze routine de waarde van de gelezen byte weer over aan de accu. In eigen programma's kunt u door tweemaal achter elkaar de instructie POP in te voeren de BASIC-pointer van de stack halen. Daarbij is het belangrijk de stack, en dan met name de terugsprongadressen, weer goed op te bouwen.

Assemblerlisting

```
F000              10                              ; slot PEEK
F000              20                              ; formaat:
                                                  ; USR1(adres),slot
F000              30                ORG  &HF000
F000 5A47         40      ILLQUA    EQU  &H475A
F002 BA2F         50      CINT      EQU  &H2FBA
F004 1C52         60      GETBYT    EQU  &H521C
F006 0C00         70      SLOTRD    EQU  &H000C
F008              80                     ;
F008 CDBA2F       90                CALL CINT     ; maakt integer en
                                                  ; laadt waarde in HL
F00B EB          100                EX   DE,HL
F00C C1          110                POP  BC       ; terugsprongadres
F00D E1          120                POP  HL       ; BASIC-pointer
F00E E5          130                PUSH HL       ; stack
F00F C5          140                PUSH BC       ; herstellen
F010 D5          150                PUSH DE       ; waarde adres
F011 CF          160                RST  &H08     ; test op
F012 2C          170                DB   &H2C     ; ASCII(",")
F013 CD1C52      180                CALL GETBYT   ; haalt slotnummer
F016 FE03        190                CP   3        ; slotnummer >= 3?
F018 D25A47      200                JP   NC, ILLQUA; ja, dan Illegal
                                                  ; quantity
F01B E3          210                EX   (SP),HL  ; verwissel BASIC-p.
                                                  ; en te lezen adres
F01C CD0C00      220                CALL SLOTRD   ; slot read
F01F CDCF4F      230                CALL &H4FCF   ; accu in FAC laden
F022 E1          240                POP  HL       ; BASIC-pointer
F023 C1          250                POP  BC       ; terugsprong
F024 D1          260                POP  DE       ; vorige BASIC-p.
F025 E5          270                PUSH HL       ; nieuwe BASIC-p.
F026 C5          280                PUSH BC       ; sprong terug
F027 21F6F7      290                LD   HL,&HF7F6 ; startadres FAC
F02A             300                              ; A bevat al type 2
                                                  ; (zie &H4FCF e.v.)
F02A C9          310                RET
--------------------------------------------------------------------
programma: SLOTRD
start: &HF000
einde: &HF02A
lengte: &H2B
0 fout
variabelentabel:
ILLQUA     475A
CINT       2F8A
```

GETBYT 521C
SLOTRD 000C

5.5 Systeemroutines

&H0000: RESET

Het resultaat van deze routine is hetzelfde als wanneer u de computer aan-/uitschakelt of op de RESET-toets drukt.

&H0008: RST &H08 - test op volgende byte

De byte op adres HL wordt vergeleken met de byte die direct op de instructie RST &H08 volgt. Indien niet gelijk: "Syntax error". Anders vertakking naar de routine RST &H10 vanaf adres &H4666 (zie aldaar).

&H000C: LD A,(HL) met slotkeuze

A bevat bij een overdracht het gewenste slotnummer (0-3). Het resultaat is dat in de accu en in register E de waarde komt die op adres HL in het gekozen slot staat.

&H0014: LD (HL),E met slotkeuze

Bij een overdracht bevat A het slotnummer, (HL) het adres en E de waarde die moet worden geschreven.

&H0018: RST &H18 - uitvoer van teken

Voert het teken in de accu uit op het actuele apparaat. De default-optie is het beeldscherm. De printer kiest u door geheugenplaats &HF416 te laden met een waarde ongelijk 0.

&H0020: RST &H20 - vergelijking HL met DE

Register DE wordt afgetrokken van register HL; de flags worden

overeenkomstig het resultaat beïnvloed. HL en DE veranderen zelf niet!

&H00028: RST &H28 - test variabeletype

Stelt het het variabeletype vast. Na overdracht gelden de volgende flags:

flag	aanduiding	typenummer	type
carry=0	(NC)	8	DBL
carry=1	(C)	2/3/4:	
sign =1	(M)	2	INT
zero =1	(Z)	3	string
sign =0	(P)	4	SNG

&H0038: RST &H38 - interruptsprong in modus 1

Vertekpunt van de sprong naar de routine die bij de standaardinterrupt 50 maal per seconde wordt aangeroepen.

&H003E: standaardfunctie van de toetsen

Deze routine geeft de functietoetsen de betekenis die ze normaal bij inschakeling hebben.

&H0047: VDP-register write

De waarde in register B gaat naar het VDP-register met het nummer dat in register C staat; dus VDP(C)=B.

&H004A: video-RAM read

Deze routine laadt de byte op adres HL van het video-RAM in de accu. BASIC: A=VPEEK(HL)

&H004D: video-RAM write

De waarde in de accu wordt opgeslagen op adres HL van het video-RAM. BASIC: VPOKE HL,A

&H005F: select SCREEN

De SCREEN-modus wordt omgeschakeld op de waarde die in de accu staat (0, 1, 2 of 3). BASIC: SCREEN A

&H0093: PSG-register write

De waarde in register E gaat naar het PSG-register met het nummer dat in A staat. Programmering van de PSG is vooral belangrijk voor de produktie van complexe geluiden. BASIC: SOUND A,E

&H0096: PSG-register read

Wordt deze routine aangeroepen, dan krijgt de accu de waarde uit het PSG-register met het nummer dat de accu daarvoor bevatte.

&H009F: op toetsdruk wachten

Door deze routine gebeurt er niets zolang er geen toets wordt ingedrukt. De ASCII-code van de ingedrukte toets wordt geregistreerd, in de accu gezet en gevolgd door een sprong terug.

&H00AE: invoer vanaf begin regel tot CR

De routine haalt een hele regel invoer terug. Een regel kan maximaal 255 tekens lang zijn en moet dus worden opgeslagen in het RAM. Daarvoor is ruimte vanaf &HF55E tot (+255) &HF65D. Na terugkeer vanuit deze routine bevat HL het startadres van de invoerbuffer minus 1.

&H00C0: BEEP

Produceert een pieptoon. BASIC: BEEP

&H00C3: clear screen

Wist in alle modi het hele bereik van het beeldschermgeheugen. BASIC: CLS

&H00C6: cursor plaatsen

De cursor komt op positie HL te staan, waarbij H de regel aangeeft en L de kolom. BASIC: LOCATE L,H

&H00CC: KEY OFF

Schakelt de standaardfunctie van een toets uit.

&H00CF: KEY ON

De toets krijgt zijn standaardfunctie terug.

&H00D5: STICK uitlezen

Na overdracht van A levert deze routine de richting die de joy-stick aangeeft aan de accu.

0	toetsenbord
1	joystick 1
2	joystick 2

BASIC: A=STICK(A)

Hoewel deze routine geheel overeenkomt met de BASIC-instructie is hij toch belangrijk in verband met de voor spelletjes noodzakelijke snelheid.

&H00D8: STRIG uitlezen

Na overdracht van A (zie boven) levert deze routine als waarde

0	als de vuurknop niet is ingedrukt
255	als de vuurknop wel is ingedrukt

&H0132: CAP aan/uit

Schakelt CAP aan of uit.

A=0	CAP aan
A<>0	CAP uit

&H0135: software sound

Deze routine maakt de lijn voor 'external sound' hoog of laag.

A=0 hoog
A<>0 laag

Deze lijn is direct verbonden met de luidspreker. Snel achter elkaar aan en uitschakelen levert een toon op. In andere gevallen blijft het bij een klik.

&H013E: VDP-status read

De actuele waarde van het VDP-statusregister wordt overgedragen aan de accu.

&H2F8A: CINT

Zet FAC om in het formaat INT (integer) en test de grootte. De waarde komt in HL en het typenummer (2) in de accu.

&H2FB2: CSNG

Zet FAC om in het formaat SNG (enkelvoudige nauwkeurigheid).

&H2F99: HL kopiëren naar FAC

Laadt de waarde van register HL in de FAC; het typenummer wordt 2.

&H303A: CDBL

Zet FAC om in het formaat DBL (dubbele nauwkeurigheid).

&H4055: Syntax error (ERR= 2)

&H4067: Overflow (ERR= 6)

&H406A: Missing operand (ERR=24)

&H406D: Type mismatch (ERR=13)

&H406F: foutmelding

Geeft de fout aan die overeenkomt met het getal in register E.

&H400B: GET PAR

Deze BASIC-interpreterroutine leest een 16-bits adres en (daarvan door een komma gescheiden) de daarop volgende 8-bits waarde. Register BC krijgt de 16-bits waarde en DE de 8-bits waarde.

&H521C: GET BYTE

Leest een 8-bits waarde die wordt teruggegeven aan de accu en aan register E.

&H542F: GET ADR

Leest een 16-bits waarde die wordt teruggegeven aan register DE.

&H5439: GET ADR tussen haakjes

Leest een 16-bits waarde die tussen haakjes staat.

5.6 Veranderingen in het systeem

Tot slot bekijken we op welke manier we het bedrijfssysteem, respectievelijk de BASIC-interpreter kunnen wijzigen. De ontwerpers van het bedrijfssysteem van de MSX hebben daartoe gebruik gemaakt van de zogenaamde patch-techniek (Engels: to patch = bedekken). Het grote probleem bij systeemveranderingen is het feit dat het ROM niet kan worden gewijzigd. Veranderingen zijn alleen mogelijk in het RAM-gedeelte. Om toch op veel plaatsen wijzigingen te kunnen aanbrengen, wordt vanuit belangrijke ROM-routines vertakt naar het RAM; dat gebeurt door een subroutinesprong die voor de eigenlijke routine staat. In de

begintoestand staat op al die sprongadressen in het RAM de code &HC9, oftewel RET. Als de betreffende adressen worden aangeroepen, gebeurt er niets; het programma keert direct terug naar het ROM en gaat zonder wijzigingen verder. Dit patch-gebied ligt in het systeem-RAM van &HFD9A tot &HFFC9. Om een verandering aan te brengen, moeten we over het betreffende sprongadres in het RAM een 'patch' leggen. De instructie RET wordt dus bedekt door de code van een andere instructie, bijvoorbeeld een spronginstructie. Bij iedere patch-sprong zijn 5 bytes beschikbaar voor een verandering. Dat is het maximum, omdat u anders al met de volgende routine in de weer bent. Hoe de patch-techniek werkt, maken we duidelijk aan de hand van het volgende voorbeeld. De opdracht is: verander de BASIC-instructie INPUT zodanig dat er geen vraagteken verschijnt. De invoerroutine begint op adres &H23CC. Begin op dat punt te vertalen met de disassembler:

```
23CC CDE0F0    CALL &HFDE0
23CF 3E3F      LD   A,&H3F
23D1 DF        RST  &H18
23D2 3E20      LD   A,&H20
23D4 DF        RST  &H18
23D5 .
 .   .
 .   .
```

De eerste instructie is de sprong naar het patch-bereik in het RAM. De adressen &HFDE0 tot &HFDE4 zijn beschikbaar om de invoerroutine te wijzigen. In het voorbeeld wordt daarna de accu geladen met de code van ?. RST &H18 voert deze uit. Het programma moet over de laatste twee instructies heenspringen en op &H23D2 verdergaan. De patch ziet er dan zo uit:

```
&HFDE0         ; startadres van de patch
POP AF         ; terugsprongadres ophalen
JP &H23D2      ; vraagteken overslaan
```

Het volgende programmaatje initialiseert de patch.

```
10 POKE &HFDE1,&HC3
20 POKE &HFDE2,&HD2
30 POKE &HFDE3,&H23
40 REM starten met:
50 POKE &HFDE0,&HF1
60 REM uitschakelen met &HFDE0,&HC9
```

Belangrijk is dat het beginadres van de patch als laatste van een nieuwe waarde wordt voorzien. Om de oude toestand te herstellen moet op die plaats weer RET (&HC9) komen te staan. Bij de

wijziging van AUTO in paragraaf 4.9 is dezelfde techniek gebruikt. Daar stond het te wijzigen bereik op grote afstand van de patchsprong. In zo'n geval is het beter eerst het hele bereik te kopiëren naar het RAM en het daar te veranderen. De in 4.9 gebruikte patch is onderdeel van de hoofdwachtlus van de computer, die in de normale invoermodus wordt doorlopen. Gelijktijdige uitvoer op beeldscherm en printer verloopt volgens hetzelfde principe. Het programma verandert de routine RST &H18 zodanig dat hij tweemaal wordt aangeroepen.

6 PERSPECTIEVEN

Zelf machinetaalprogramma's maken moet zo langzamerhand niet al te veel problemen meer opleveren. Voor programma's van enige omvang is programmeren in assembler onvermijdelijk. Dat kost meer tijd dan werken met een hogere taal. Conclusie: goede hulpprogramma's zijn een eerste vereiste. Om zelf machinetaalprogramma's te schrijven hebt u minstens een assemblerprogramma en een omvangrijk monitorprogramma nodig. Met bijvoorbeeld de pseudo-instructies kunt u het programmeren nog verder vereenvoudigen. Macro's, voorwaardelijke assemblage en externe programma's en variabelen bewijzen in dit opzicht ook goede diensten.

Macro's

Vaak komt een bepaalde instructiereeks meerdere keren in een programma voor. Door gebruik te maken van een macro, die in het source-programma de instructiereeks vervangt, hoeft u die reeks niet steeds opnieuw in te voeren. De assembler kan macro's onderscheiden en weet dat het om een afkorting gaat. Macro's maken source-programma's overzichtelijker en korter.

Voorwaardelijke assemblage

Door een voorwaarde te gebruiken geeft u aan welke delen van een programma moeten worden vertaald. Een algemeen source-programma kan op die manier worden geschreven als programma voor bestandsbeheer en worden toegesneden op een bepaalde functie.

Externe programma's en variabelen

Programmeren in assembler moet gestructureerd gebeuren. Verdeel grotere vraagstukken in kleinere en schrijf elk programma-onderdeel apart. Sommige subroutines worden vaak gebruikt, bijvoorbeeld de routine die een teken hexadecimaal uitvoert. Bij een handige assembler zijn dat soort routines en variabelen gerubriceerd. Die routines worden opgeroepen door ze bij hun naam te noemen. Treft de computer zo'n naam aan in het source-programma dan wordt de routine van band of schijf geladen en ingevoegd in het objectprogramma. Het programma dat verschillende machinetaalprogramma's verbindt, heet LINKER (to link). Meestal hoort daar een RELOCATOR (verschuiver) bij. Deze

corrigeert adressen die veranderd zijn doordat programa's zijn ingevoegd of verschoven. Programma's met dit snufje zijn meestal omvangrijk en kostbaar maar wel snel en handig. Veel assemblers hebben een eigen editor; de invoer van assemblerinstructies is dan niet meer aan een regelnummer gebonden. Bij een assembler horen een aantal hulpprogramma's. De meeste daarvan worden ondergebracht in een monitor. Ook de disassembler maakt vaak deel uit van het monitorprogramma. Machinetaalprogramma's testen is een heel belangrijke monitorfunctie; een breakpoint zetten is de eenvoudigste manier. Een DEBUGGER ("foutendetector") is veel gecompliceerder. Deze faciliteit bestaat uit een aantal testroutines met als belangrijkste component de stap-voor-stap simulator.

Zo kunnen we nog wel een tijdje doorgaan. Ter ontnuchtering: met hulpprogramma's alleen komt u er niet. Programmeren krijgt u pas in de vingers na veel oefening. De basis voor de programmering van de Z80 is in de vorige hoofdstukken gelegd. Veel plezier bij de realisatie van uw eigen ideëen.

Bijlage 1

Conversietabel decimaal - hexadecimaal - binair

decimaal	hex	binair	decimaal	hex	binair
0	&H00	&B00000000	26	&H1A	&B00011010
1	&H01	&B00000001	27	&H1B	&B00011011
2	&H02	&B00000010	28	&H1C	&B00011100
3	&H03	&B00000011	29	&H1D	&B00011101
4	&H04	&B00000100	30	&H1E	&B00011110
5	&H05	&B00000101	31	&H1F	&B00011111
6	&H06	&B00000110	32	&H20	&B00100000
7	&H07	&B00000111	33	&H21	&B00100001
8	&H08	&B00001000	34	&H22	&B00100010
9	&H09	&B00001001	35	&H23	&B00100011
10	&H0A	&B00001010	36	&H24	&B00100100
11	&H0B	&B00001011	37	&H25	&B00100101
12	&H0C	&B00001100	38	&H26	&B00100110
13	&H0D	&B00001101	39	&H27	&B00100111
14	&H0E	&B00001110	40	&H28	&B00101000
15	&H0F	&B00001111	41	&H29	&B00101001
16	&H10	&B00010000	42	&H2A	&B00101010
17	&H11	&B00010001	43	&H2B	&B00101011
18	&H12	&B00010010	44	&H2C	&B00101100
19	&H13	&B00010011	45	&H2D	&B00101101
20	&H14	&B00010100	46	&H2E	&B00101110
21	&H15	&B00010101	47	&H2F	&B00101111
22	&H16	&B00010110	48	&H30	&B00110000
23	&H17	&B00010111	49	&H31	&B00110001
24	&H18	&B00011000	50	&H32	&B00110010
25	&H19	&B00011001	51	&H33	&B00110011

decimaal	hex	binair	decimaal	hex	binair
52	&H34	&B00110100	78	&H4E	&B01001110
53	&H35	&B00110101	79	&H4F	&B01001111
54	&H36	&B00110110	80	&H50	&B01010000
55	&H37	&B00110111	81	&H51	&B01010001
56	&H38	&B00111000	82	&H52	&B01010010
57	&H39	&B00111001	83	&H53	&B01010011
58	&H3A	&B00111010	84	&H54	&B01010100
59	&H3B	&B00111011	85	&H55	&B01010101
60	&H3C	&B00111100	86	&H56	&B01010110
61	&H3D	&B00111101	87	&H57	&B01010111
62	&H3E	&B00111110	88	&H58	&B01011000
63	&H3F	&B00111111	89	&H59	&B01011001
64	&H40	&B01000000	90	&H5A	&B01011010
65	&H41	&B01000001	91	&H5B	&B01011011
66	&H42	&B01000010	92	&H5C	&B01011100
67	&H43	&B01000011	93	&H5D	&B01011101
68	&H44	&B01000100	94	&H5E	&B01011110
69	&H45	&B01000101	95	&H5F	&B01011111
70	&H46	&B01000110	96	&H60	&B01100000
71	&H47	&B01000111	97	&H61	&B01100001
72	&H48	&B01001000	98	&H62	&B01100010
73	&H49	&B01001001	99	&H63	&B01100011
74	&H4A	&B01001010	100	&H64	&B01100100
75	&H4B	&B01001011	101	&H65	&B01100101
76	&H4C	&B01001100	102	&H66	&B01100110
77	&H4D	&B01001101	103	&H67	&B01100111

decimaal	hex	binair	decimaal	hex	binair
104	&H68	&B01101000	130	&H82	&B10000010
105	&H69	&B01101001	131	&H83	&B10000011
106	&H6A	&B01101010	132	&H84	&B10000100
107	&H6B	&B01101011	133	&H85	&B10000101
108	&H6C	&B01101100	134	&H86	&B10000110
109	&H6D	&B01101101	135	&H87	&B10000111
110	&H6E	&B01101110	136	&H88	&B10001000
111	&H6F	&B01101111	137	&H89	&B10001001
112	&H70	&B01110000	138	&H8A	&B10001010
113	&H71	&B01110001	139	&H8B	&B10001011
114	&H72	&B01110010	140	&H8C	&B10001100
115	&H73	&B01110011	141	&H8D	&B10001101
116	&H74	&B01110100	142	&H8E	&B10001110
117	&H75	&B01110101	143	&H8F	&B10001111
118	&H76	&B01110110	144	&H90	&B10010000
119	&H77	&B01110111	145	&H91	&B10010001
120	&H78	&B01111000	146	&H92	&B10010010
121	&H79	&B01111001	147	&H93	&B10010011
122	&H7A	&B01111010	148	&H94	&B10010100
123	&H7B	&B01111011	149	&H95	&B10010101
124	&H7C	&B01111100	150	&H96	&B10010110
125	&H7D	&B01111101	151	&H97	&B10010111
126	&H7E	&B01111110	152	&H98	&B10011000
127	&H7F	&B01111111	153	&H99	&B10011001
128	&H80	&B10000000	154	&H9A	&B10011010
129	&H81	&B10000001	155	&H9B	&B10011011

decimaal	hex	binair	decimaal	hex	binair
156	&H9C	&B10011100	182	&HB6	&B10110110
157	&H9D	&B10011101	183	&HB7	&B10110111
158	&H9E	&B10011110	184	&HB8	&B10111000
159	&H9F	&B10011111	185	&HB9	&B10111001
160	&HA0	&B10100000	186	&HBA	&B10111010
161	&HA1	&B10100001	187	&HBB	&B10111011
162	&HA2	&B10100010	188	&HBC	&B10111100
163	&HA3	&B10100011	189	&HBD	&B10111101
164	&HA4	&B10100100	190	&HBE	&B10111110
165	&HA5	&B10100101	191	&HBF	&B10111111
166	&HA6	&B10100110	192	&HC0	&B11000000
167	&HA7	&B10100111	193	&HC1	&B11000001
168	&HA8	&B10101000	194	&HC2	&B11000010
169	&HA9	&B10101001	195	&HC3	&B11000011
170	&HAA	&B10101010	196	&HC4	&B11000100
171	&HAB	&B10101011	197	&HC5	&B11000101
172	&HAC	&B10101100	198	&HC6	&B11000110
173	&HAD	&B10101101	199	&HC7	&B11000111
174	&HAE	&B10101110	200	&HC8	&B11001000
175	&HAF	&B10101111	201	&HC9	&B11001001
176	&HB0	&B10110000	202	&HCA	&B11001010
177	&HB1	&B10110001	203	&HCB	&B11001011
178	&HB2	&B10110010	204	&HCC	&B11001100
179	&HB3	&B10110011	205	&HCD	&B11001101
180	&HB4	&B10110100	206	&HCE	&B11001110
181	&HB5	&B10110101	207	&HCF	&B11001111

decimaal	hex	binair	decimaal	hex	binair
208	&HD0	&B11010000	234	&HEA	&B11101010
209	&HD1	&B11010001	235	&HEB	&B11101011
210	&HD2	&B11010010	236	&HEC	&B11101100
211	&HD3	&B11010011	237	&HED	&B11101101
212	&HD4	&B11010100	238	&HEE	&B11101110
213	&HD5	&B11010101	239	&HEF	&B11101111
214	&HD6	&B11010110	240	&HF0	&B11110000
215	&HD7	&B11010111	241	&HF1	&B11110001
216	&HD8	&B11011000	242	&HF2	&B11110010
217	&HD9	&B11011001	243	&HF3	&B11110011
218	&HDA	&B11011010	244	&HF4	&B11110100
219	&HDB	&B11011011	245	&HF5	&B11110101
220	&HDC	&B11011100	246	&HF6	&B11110110
221	&HDD	&B11011101	247	&HF7	&B11110111
222	&HDE	&B11011110	248	&HF8	&B11111000
223	&HDF	&B11011111	249	&HF9	&B11111001
224	&HE0	&B11100000	250	&HFA	&B11111010
225	&HE1	&B11100001	251	&HFB	&B11111011
226	&HE2	&B11100010	252	&HFC	&B11111100
227	&HE3	&B11100011	253	&HFD	&B11111101
228	&HE4	&B11100100	254	&HFE	&B11111110
229	&HE5	&B11100101	255	&HFF	&B11111111
230	&HE6	&B11100110			
231	&HE7	&B11100111			
232	&HE8	&B11101000			
233	&HE9	&B11101001			

Bijlage 2

Uitleg bij de instructietabellen

In de eerste tabel staan pijlen voor de codes &HCB, &HED, &HDD en &HFD. Dit betekent:

&HCB

Is &HCB de eerste code die moet worden vertaald dan moet de tweede code in de tweede tabel worden opgezocht. Het gaat hier om roteer- en schuifinstructies.

&HED

Is &HED de eerste code die moet worden vertaald dan moet de tweede code in de derde tabel worden opgezocht.

&HDD en **&HFD**

Is de eerste code &HDD of &HFD dan gaat het om geïndiceerd geadresseerde instructies. &HDD betreft het IX-register, &HFD het IY-register. Deze instructies komen niet in een aparte tabel voor. U kunt ze op de volgende manier afleiden: zoek de tweede code op in de tabel. De verkregen instructie moet het HL-register bevatten. Komt dit niet voor in de operand of is het resultaat EX DE,HL, dan gaat het om een ongeldige instructie. In dat geval voert de disassembler ??? uit. Bij een geldige instructie moet het register HL worden vervangen door IX of IY.

HL wordt IX of IY
HL wordt (IX+d) of (IY+d);
d komt overeen met de 3e code

Deze regels gelden met uitzondering van JP (HL) voor alle instructies die HL bevatten. Ondanks dat in JP (HL) register HL tussen haakjes staat, wordt het resultaat toch JP (IX) of JP (IY) na invoeging van de indexregisters.

Bijlage 3

Instructietabellen

	0	1	2	3	4	5	6	7
0	NOP	LD BC,nn	LD (BC),A	INC BC	INC B	DEC B	LD B,n	RLCA
1	DJNZ of	LD DE,nn	LD (DE),A	INC DE	INC D	DEC D	LD D,n	RLA
2	JR NZ,of	LD HL,nn	LD (nn),HL	INC HL	INC H	DEC H	LD H,n	DAA
3	JR NC,of	LD SP,nn	LD (nn),A	INC SP	INC (HL)	DEC (HL)	LD (HL),n	SCF
4	LD B,B	LD B,C	LD B,D	LD B,E	LD B,H	LD B,L	LD B,(HL)	LD B,A
5	LD D,B	LD D,C	LD D,D	LD D,E	LD D,H	LD D,L	LD D,(HL)	LD D,A
6	LD H,B	LD H,C	LD H,D	LD H,E	LD H,H	LD H,L	LD H,(HL)	LD H,A
7	LD (HL),B	LD (HL),C	LD (HL),D	LD (HL),E	LD (HL),H	LD (HL),L	HALT	LD (HL),A
8	ADD A,B	ADD A,C	ADD A,D	ADD A,E	ADD A,H	ADD A,L	ADD A,(HL)	ADD A,A
9	SUB B	SUB C	SUB D	SUB E	SUB H	SUB L	SUB (HL)	SUB A
A	AND B	AND C	AND D	AND E	AND H	AND L	AND (HL)	AND A
B	OR B	OR C	OR D	OR E	OR H	OR L	OR (HL)	OR A
C	RET NZ	POP BC	JP NZ,nn	JP nn	CALL NZ,nn	PUSH BC	ADD A,n	RST & 00
D	RET NC	POP DE	JP NC,nn	OUT (n),A	CALL NC,nn	PUSH DE	SUB n	RST & 10
E	RET PO	POP HL	JP PO,nn	EX (SP),HL	CALL PO,nn	PUSH HL	AND n	RST &20
F	RET P	POP AF	JP P,nn	DI	CALL P,nn	PUSH AF	OR n	RST & 30

	8	9	A	B	C	D	E	F
0	EX AF,AF	ADD HL,BC	LD A,(BC)	DEC BC	INC C	DEC C	LD C,n	RRCA
1	JR of	ADD HL,DE	LD A,(DE)	DEC DE	INC E	DEC E	LD E,n	RRA
2	JR Z,of	ADD HL,HL	LD HL,(nn)	DEC HL	INC L	DEC L	LD L,n	CPL
3	JR C,of	ADD HL,SP	LD A,(nn)	DEC SP	INC A	DEC A	LD A,n	CCF
4	LD C,B	LD C,C	LD C,D	LD C,E	LD C,H	LD C,L	LD C,(HL)	LD C,A
5	LD E,B	LD E,C	LD E,D	LD E,E	LD E,H	LD E,L	LD E,(HL)	LD E,A
6	LD L,B	LD L,C	LD L,D	LD L,E	LD L,H	LD L,L	LD L,(HL)	LD L,A
7	LD A,B	LD A,C	LD A,D	LD A,E	LD A,H	LD A,L	LD A,(HL)	LD A,A
8	ADC A,B	ADC A,C	ADC A,D	ADC A,E	ADC A,H	ADC A,L	ADC A,(HL)	ADC A,A
9	SBC A,B	SBC A,C	SBC A,D	SBC A,E	SBC A,H	SBC A,L	SBC A,(HL)	SBC A,A
A	XOR B	XOR C	XOR D	XOR E	XOR H	XOR L	XOR (HL)	XOR A
B	CP B	CP C	CP D	CP E	CP H	CP L	CP (HL)	CP A
C	RET Z	RET	JP Z,nn	⟶	CALL Z,nn	CALL nn	ADC A,n	RST & 08
D	RET C	EXX	JP C,nn	IN A,(n)	CALL C,nn	⟶	SBC A,n	RST & 18
E	RET PE	JP (HL)	JP PE,nn	EX DE,HL	CALL PE,nn	⟶	XOR n	RST & 28
F	RET M	LD SP,HL	JP M,nn	EI	CALL M,nn	⟶	CP n	RST & 38

	0	1	2	3	4	5	6	7
0	RLC B	RLC C	RLC D	RLC E	RLC H	RLC L	RLC (HL)	RLC A
1	RL B	RL C	RL D	RL E	RL H	RL L	RL (HL)	RL A
2	SLA B	SLA C	SLA D	SLA E	SLA H	SLA L	SLA (HL)	SLA A
3								
4	BIT 0,B	BIT 0,C	BIT 0,D	BIT 0,E	BIT 0,H	BIT 0,L	BIT 0,(HL)	BIT 0,A
5	BIT 2,B	BIT 2,C	BIT 2,D	BIT 2,E	BIT 2,H	BIT 2,L	BIT 2,(HL)	BIT 2,A
6	BIT 4,B	BIT 4,C	BIT 4,D	BIT 4,E	BIT 4,H	BIT 4,L	BIT 4,(HL)	BIT 4,A
7	BIT 6,B	BIT 6,C	BIT 6,D	BIT 6,E	BIT 6,H	BIT 6,L	BIT 6,(HL)	BIT 6,A
8	RES 0,B	RES 0,C	RES 0,D	RES 0,E	RES 0,H	RES 0,L	RES 0,(HL)	RES 0,A
9	RES 2,B	RES 2,C	RES 2,D	RES 2,E	RES 2,H	RES 2,L	RES 2,(HL)	RES 2,A
A	RES 4,B	RES 4,C	RES 4,D	RES 4,E	RES 4,H	RES 4,L	RES 4,(HL)	RES 4,A
B	RES 6,B	RES 6,C	RES 6,D	RES 6,E	RES 6,H	RES 6,L	RES 6,(HL)	RES 6,A
C	SET 0,B	SET 0,C	SET 0,D	SET 0,E	SET 0,H	SET 0,L	SET 0,(HL)	SET 0,A
D	SET 2,B	SET 2,C	SET 2,D	SET 2,E	SET 2,H	SET 2,L	SET 2,(HL)	SET 2,A
E	SET 4,B	SET 4,C	SET 4,D	SET 4,E	SET 4,H	SET 4,L	SET 4,(HL)	SET 4,A
F	SET 6,B	SET 6,C	SET 6,D	SET 6,E	SET 6,H	SET 6,L	SET 6,(HL)	SET 6,A

	8	9	A	B	C	D	E	F
0	RRC B	RRC C	RRC D	RRC E	RRC H	RRC L	RRC (HL)	RRC A
1	RR B	RR C	RR D	RR E	RR H	RR L	RR (HL)	RR A
2	SRA B	SRA C	SRA D	SRA E	SRA H	SRA L	SRA (HL)	SRA A
3	SRL B	SRL C	SRL D	SRL E	SRL H	SRL L	SRL (HL)	SRL A
4	BIT 1,B	BIT 1,C	BIT 1,D	BIT 1,E	BIT 1,H	BIT 1,L	BIT 1,(HL)	BIT 1,A
5	BIT 3,B	BIT 3,C	BIT 3,D	BIT 3,E	BIT 3,H	BIT 3,L	BIT 3,(HL)	BIT 3,A
6	BIT 5,B	BIT 5,C	BIT 5,D	BIT 5,E	BIT 5,H	BIT 5,L	BIT 5,(HL)	BIT 5,A
7	BIT 7,B	BIT 7,C	BIT 7,D	BIT 7,E	BIT 7,H	BIT 7,L	BIT 7,(HL)	BIT 7,A
8	RES 1,B	RES 1,C	RES 1,D	RES 1,E	RES 1,H	RES 1,L	RES 1,(HL)	RES 1,A
9	RES 3,B	RES 3,C	RES 3,D	RES 3,E	RES 3,H	RES 3,L	RES 3,(HL)	RES 3,A
A	RES 5,B	RES 5,C	RES 5,D	RES 5,E	RES 5,H	RES 5,L	RES 5,(HL)	RES 5,A
B	RES 7,B	RES 7,C	RES 7,D	RES 7,E	RES 7,H	RES 7,L	RES 7,(HL)	RES 7,A
C	SET 1,B	SET 1,C	SET 1,D	SET 1,E	SET 1,H	SET 1,L	SET 1,(HL)	SET 1,A
D	SET 3,B	SET 3,C	SET 3,D	SET 3,E	SET 3,H	SET 3,L	SET 3,(HL)	SET 3,A
E	SET 5,B	SET 5,C	SET 5,D	SET 5,E	SET 5,H	SET 5,L	SET 5,(HL)	SET 5,A
F	SET 7,B	SET 7,C	SET 7,D	SET 7,E	SET 7,H	SET 7,L	SET 7,(HL)	SET 7,A

	0	1	2	3	4	5	6	7
4	IN B,(C)	OUT (C),B	SBC HL,BC	LD (nn),BC	NEG	RETN	IM 0	LD I,A
5	IN D,(C)	OUT (C),P	SBC HL,DE	LD (nn),DE			IM 1	LD A,I
6	IN H,(C)	OUT (C),H	SBC HL,HL	LD (nn),HL				RRD
7			SBC HL,SP	LD (nn),SP				
8								
9								
A	LDI	CPI	INI	OUTI				
	LDIR	CPIR	INIR	OTIR				

	8	9	A	B	C	D	E	F
4	IN C,(C)	OUT (C),C	ADC HL,BC	LD BC,(nn)		RETI		LD R,A
5	IN E,(C)	OUT (C),E	ADC HL,DE	LD DE,(nn)			IM 2	LD A,R
6	IN L,(C)	OUT (C),L	ADC HL,HL	LD HL,(nn)				RLD
7	IN A,(C)	OUT (C),A	ADC HL,SP	LD SP,(nn)				
8								
9								
A	LDD	CPD	IND	OUTD				
B	LDDR	CPDR	INDR	OTDR				

Bijlage 4

Zilog instructietabellen

		SOURCE															
		IMPLIED		REGISTER							REG INDIRECT			INDEXED		EXT. ADDR.	IMME.
		I	R	A	B	C	D	E	H	L	(HL)	(BC)	(DE)	(IX + d)	(IY + d)	(nn)	n
DESTINATION REGISTER	A	ED 57	ED 5F	7F	78	79	7A	7B	7C	7D	7E	0A	1A	DD 7E d	FD 7E d	3A n n	3E n
	B			47	40	41	42	43	44	45	46			DD 46 d	FD 46 d		06 n
	C			4F	48	49	4A	4B	4C	4D	4E			DD 4E d	FD 4E d		0E n
	D			57	50	51	52	53	54	55	56			DD 56 d	FD 56 d		16 n
	E			5F	58	59	5A	5B	5C	5D	5E			DD 5E d	FD 5E d		1E n
	H			67	60	61	62	63	64	65	66			DD 66 d	FD 66 d		26 n
	L			6F	68	69	6A	6B	6C	6D	6E			DD 6E d	FD 6E d		2E n
REG INDIRECT	(HL)			77	70	71	72	73	74	75							36 n
	(BC)			02													
	(DE)			12													
INDEXED	(IX+d)			DD 77 d	DD 70 d	DD 71 d	DD 72 d	DD 73 d	DD 74 d	DD 75 d							DD 36 d n
	(IY+d)			FD 77 d	FD 70 d	FD 71 d	FD 72 d	FD 73 d	FD 74 d	FD 75 d							FD 36 d n
EXT. ADDR	(nn)			32 n n													
IMPLIED	I			ED 47													
	R			ED 4F													

		SOURCE									
		REGISTER							IMM. EXT.	EXT. ADDR.	REG. INDIR.
		AF	BC	DE	HL	SP	IX	IY	nn	(nn)	(SP)
DESTINATION REGISTER	AF										F1
	BC								01 n n	ED 4B n n	C1
	DE								11 n n	ED 5B n n	D1
	HL								21 n n	2A n n	E1
	SP				F9		DD F9	FD F9	31 n n	ED 7B n n	
	IX								DD 21 n n	DD 2A n n	DD E1
	IY								FD 21 n n	FD 2A n n	FD E1
EXT. ADDR.	(nn)		ED 43 n n	ED 53 n n	22 n n	ED 73 n n	DD 22 n n	FD 22 n n			
PUSH INSTRUCTIONS → REG. IND.	(SP)	F5	C5	D5	E5		DD E5	FD E5			

↑ POP INSTRUCTIONS

NOTE: The Push & Pop Instructions adjust the SP after every execution

		IMPLIED ADDRESSING				
		AF′	BC′, DE′ & HL′	HL	IX	IY
IMPLIED	AF	08				
	BC, DE & HL		D9			
	DE			EB		
REG. INDIR.	(SP)			E3	DD E3	FD E3

			SOURCE	
			REG. INDIR.	
			(HL)	
DESTINATION	REG. INDIR.	(DE)	ED A0	'LDI' – Load (DE)←(HL) Inc HL & DE, Dec BC
			ED B0	'LDIR,' – Load (DE)←(HL) Inc HL & DE, Dec BC, Repeat until BC = 0
			ED A8	'LDD' – Load (DE)←(HL) Dec HL & DE, Dec BC
			ED B8	'LDDR' – Load (DE)←(HL) Dec HL & DE, Dec BC, Repeat until BC = 0

Reg HL points to source
Reg DE points to destination
Reg BC is byte counter

SEARCH LOCATION	
REG. INDIR.	
(HL)	
ED A1	'CPI' Inc HL, Dec BC
ED B1	'CPIR', Inc HL, Dec BC repeat until BC = 0 or find match
ED A9	'CPD' Dec HL & BC
ED B9	'CPDR' Dec HL & BC Repeat until BC = 0 or find match

HL points to location in memory
to be compared with accumulator
contents
BC is byte counter

SOURCE

	REGISTER ADDRESSING							REG. INDIR.	INDEXED		IMMED.
	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)	n
'ADD'	87	80	81	82	83	84	85	86	DD 86 d	FD 86 d	C6 n
ADD w CARRY 'ADC'	8F	88	89	8A	8B	8C	8D	8E	DD 8E d	FD 8E d	CE n
SUBTRACT 'SUB'	97	90	91	92	93	94	95	96	DD 96 d	FD 96 d	D6 n
SUB w CARRY 'SBC'	9F	98	99	9A	9B	9C	9D	9E	DD 9E d	FD 9E d	DE n
'AND'	A7	A0	A1	A2	A3	A4	A5	A6	DD A6 d	FD A6 d	E6 n
'XOR'	AF	A8	A9	AA	AB	AC	AD	AE	DD AE d	FD AE d	EE n
'OR'	B7	B0	B1	B2	B3	B4	B5	B6	DD B6 d	FD B6 d	F6 n
COMPARE 'CP'	BF	B8	B9	BA	BB	BC	BD	BE	DD BE d	FD BE d	FE n
INCREMENT 'INC'	3C	04	0C	14	1C	24	2C	34	DD 34 d	FD 34 d	
DECREMENT 'DEC'	3D	05	0D	15	1D	25	2D	35	DD 35 d	FD 35 d	

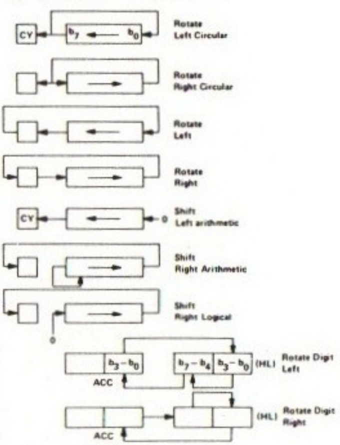
Decimal Adjust Acc, 'DAA'	27
Complement Acc, 'CPL'	2F
Negate Acc, 'NEG' (2's complement)	ED 44
Complement Carry Flag, 'CCF'	3F
Set Carry Flag, 'SCF'	37

			SOURCE					
			BC	DE	HL	SP	IX	IY
DESTINATION	'ADD'	HL	09	19	29	39		
		IX	DD 09	DD 19		DD 39	DD 29	
		IY	FD 09	FD 19		FD 39		FD 29
	ADD WITH CARRY AND SET FLAGS 'ADC'	HL	ED 4A	ED 5A	ED 6A	ED 7A		
	SUB WITH CARRY AND SET FLAGS 'SBC'	HL	ED 42	ED 52	ED 62	ED 72		
	INCREMENT 'INC'		03	13	23	33	DD 23	FD 23
	DECREMENT 'DEC'		0B	1B	2B	3B	DD 2B	FD 2B

Source and Destination

TYPE OF ROTATE OR SHIFT		A	B	C	D	E	H	L	(HL)	(IX + d)	(IY + d)
	'RLC'	CB 07	CB 00	CB 01	CB 02	CB 03	CB 04	CB 05	CB 06	DD CB d 06	FD CB d 06
	'RRC'	CB 0F	CB 08	CB 09	CB 0A	CB 0B	CB 0C	CB 0D	CB 0E	DD CB d 0E	FD CB d 0E
	'RL'	CB 17	CB 10	CB 11	CB 12	CB 13	CB 14	CB 15	CB 16	DD CB d 16	FD CB d 16
	'RR'	CB 1F	CB 18	CB 19	CB 1A	CB 1B	CB 1C	CB 1D	CB 1E	DD CB d 1E	FD CB d 1E
	'SLA'	CB 27	CB 20	CB 21	CB 22	CB 23	CB 24	CB 25	CB 26	DD CB d 26	FD CB d 26
	'SRA'	CB 2F	CB 28	CB 29	CB 2A	CB 2B	CB 2C	CB 2D	CB 2E	DD CB d 2E	FD CB d 2E
	'SRL'	CB 3F	CB 38	CB 39	CB 3A	CB 3B	CB 3C	CB 3D	CB 3E	DD CB d 3E	FD CB d 3E
	'RLD'								ED 6F		
	'RRD'								ED 67		

	A
RLCA	07
RRCA	0F
RLA	17
RRA	1F



		REGISTER ADDRESSING							REG. INDIR.	INDEXED	
	BIT	A	B	C	D	E	H	L	(HL)	(IX+d)	(IY+d)
TEST 'BIT'	0	CB 47	CB 40	CB 41	CB 42	CB 43	CB 44	CB 45	CB 46	DD CB d 46	FD CB d 46
	1	CB 4F	CB 48	CB 49	CB 4A	CB 4B	CB 4C	CB 4D	CB 4E	DD CB d 4E	FD CB d 4E
	2	CB 57	CB 50	CB 51	CB 52	CB 53	CB 54	CB 55	CB 56	DD CB d 56	FD CB d 56
	3	CB 5F	CB 58	CB 59	CB 5A	CB 5B	CB 5C	CB 5D	CB 5E	DD CB d 5E	FD CB d 5E
	4	CB 67	CB 60	CB 61	CB 62	CB 63	CB 64	CB 65	CB 66	DD CB d 66	FD CB d 66
	5	CB 6F	CB 68	CB 69	CB 6A	CB 6B	CB 6C	CB 6D	CB 6E	DD CB d 6E	FD CB d 6E
	6	CB 77	CB 70	CB 71	CB 72	CB 73	CB 74	CB 75	CB 76	DD CB d 76	FD CB d 76
	7	CB 7F	CB 78	CB 79	CB 7A	CB 7B	CB 7C	CB 7D	CB 7E	DD CB d 7E	FD CB d 7E
RESET BIT 'RES'	0	CB 87	CB 80	CB 81	CB 82	CB 83	CB 84	CB 85	CB 86	DD CB d 86	FD CB d 86
	1	CB 8F	CB 88	CB 89	CB 8A	CB 8B	CB 8C	CB 8D	CB 8E	DD CB d 8E	FD CB d 8E
	2	CB 97	CB 90	CB 91	CB 92	CB 93	CB 94	CB 95	CB 96	DD CB d 96	FD CB d 96
	3	CB 9F	CB 98	CB 99	CB 9A	CB 9B	CB 9C	CB 9D	CB 9E	DD CB d 9E	FD CB d 9E
	4	CB A7	CB A0	CB A1	CB A2	CB A3	CB A4	CB A5	CB A6	DD CB d A6	FD CB d A6
	5	CB AF	CB A8	CB A9	CB AA	CB AB	CB AC	CB AD	CB AE	DD CB d AE	FD CB d AE
	6	CB B7	CB B0	CB B1	CB B2	CB B3	CB B4	CB B5	CB B6	DD CB d B6	FD CB d B6
	7	CB BF	CB B8	CB B9	CB BA	CB BB	CB BC	CB BD	CB BE	DD CB d BE	FD CB d BE
SET BIT 'SET'	0	CB C7	CB C0	CB C1	CB C2	CB C3	CB C4	CB C5	CB C6	DD CB d C6	FD CB d C6
	1	CB CF	CB C8	CB C9	CB CA	CB CB	CB CC	CB CD	CB CE	DD CB d CE	FD CB d CE
	2	CB D7	CB D0	CB D1	CB D2	CB D3	CB D4	CB D5	CB D6	DD CB d D6	FD CB d D6
	3	CB DF	CB D8	CB D9	CB DA	CB DB	CB DC	CB DD	CB DE	DD CB d DE	FD CB d DE
	4	CB E7	CB E0	CB E1	CB E2	CB E3	CB E4	CB E5	CB E6	DD CB d E6	FD CB d E6
	5	CB EF	CB E8	CB E9	CB EA	CB EB	CB EC	CB ED	CB EE	DD CB d EE	FD CB d EE
	6	CB F7	CB F0	CB F1	CB F2	CB F3	CB F4	CB F5	CB F6	DD CB d F6	FD CB d F6
	7	CB FF	CB F8	CB F9	CB FA	CB FB	CB FC	CB FD	CB FE	DD CB d FE	FD CB d FE

CONDITION

			UN-COND.	CARRY	NON CARRY	ZERO	NON ZERO	PARITY EVEN	PARITY ODD	SIGN NEG	SIGN POS	REG B≠0
JUMP 'JP'	IMMED. EXT.	nn	C3 n n	DA n n	D2 n n	CA n n	C2 n n	EA n n	E2 n n	FA n n	F2 n n	
JUMP 'JR'	RELATIVE	PC+e	18 e-2	38 e-2	30 e-2	28 e-2	20 e-2					
JUMP 'JP'	REG. INDIR.	(HL)	E9									
JUMP 'JP'	REG. INDIR.	(IX)	DD E9									
JUMP 'JP'	REG. INDIR.	(IY)	FD E9									
'CALL'	IMMED. EXT.	nn	CD n n	DC n n	D4 n n	CC n n	C4 n n	EC n n	E4 n n	FC n n	F4 n n	
DECREMENT B, JUMP IF NON ZERO 'DJNZ'	RELATIVE	PC+e										10 e-2
RETURN 'RET'	REGISTER INDIR.	(SP) (SP+1)	C9	D8	D0	C8	C0	E8	E0	F8	F0	
RETURN FROM INT 'RETI'	REG. INDIR.	(SP) (SP+1)	ED 4D									
RETURN FROM NON MASKABLE INT 'RETN'	REG. INDIR.	(SP) (SP+1)	ED 45									

NOTE—CERTAIN FLAGS HAVE MORE THAN ONE PURPOSE. REFER TO SECTION 6.0 FOR DETAILS

SOURCE

			REGISTER							REG. IND.	
			A	B	C	D	E	H	L	(HL)	
'OUT'	IMMED.	(n)	D3 n								
'OUT'	REG. IND.	(C)	ED 79	ED 41	ED 49	ED 51	ED 59	ED 61	ED 69		
'OUTI' – OUTPUT Inc HL, Dec b	REG. IND.	(C)								ED A3	BLOCK OUTPUT COMMANDS
'OTIR' – OUTPUT, Inc HL, Dec B, REPEAT IF B≠0	REG. IND.	(C)								ED B3	BLOCK OUTPUT COMMANDS
'OUTD' – OUTPUT Dec HL & B	REG. IND.	(C)								ED AB	BLOCK OUTPUT COMMANDS
'OTDR' – OUTPUT, Dec HL & B, REPEAT IF B≠0	REG. IND.	(C)								ED BB	BLOCK OUTPUT COMMANDS

PORT DESTINATION ADDRESS

			SOURCE PORT ADDRESS: IMMED. (n)	SOURCE PORT ADDRESS: REG. INDIR. (C)	
INPUT DESTINATION: INPUT 'IN'	REG ADDRESSING	A	DB n	ED 78	
		B		ED 40	
		C		ED 48	
		D		ED 50	
		E		ED 58	
		H		ED 60	
		L		ED 68	
'INI' – INPUT & Inc HL, Dec B	REG. INDIR	(HL)		ED A2	BLOCK INPUT COMMANDS
'INIR' – INP, Inc HL, Dec B, REPEAT IF B≠0	REG. INDIR	(HL)		ED B2	BLOCK INPUT COMMANDS
'IND' – INPUT & Dec HL, Dec B	REG. INDIR	(HL)		ED AA	BLOCK INPUT COMMANDS
'INDR' – INPUT, Dec HL, Dec B, REPEAT IF B≠0	REG. INDIR	(HL)		ED BA	BLOCK INPUT COMMANDS

CALL ADDRESS		OP CODE	
	0000_H	C7	'RST 0'
	0008_H	CF	'RST 8'
	0010_H	D7	'RST 16'
	0018_H	DF	'RST 24'
	0020_H	E7	'RST 32'
	0028_H	EF	'RST 40'
	0030_H	F7	'RST 48'
	0038_H	FF	'RST 56'

'NOP'	00	
'HALT'	76	
DISABLE INT '(DI)'	F3	
ENABLE INT '(EI)'	FB	
SET INT MODE 0 'IM0'	ED 46	8080A MODE
SET INT MODE 1 'IM1'	ED 56	CALL TO LOCATION 0038_H.
SET INT MODE 2 'IM2'	ED 5E	INDIRECT CALL USING REGISTER I AND 8 BITS FROM INTERRUPTING DEVICE AS A POINTER.

Bijlage 5

Flag-beïnvloedingstabel

Instruction	C	Z	P/V	S	N	H	Comments
ADD A, s; ADC A,s	↕	↕	V	↕	0	↕	8-bit add or add with carry
SUB s; SBC A, s, CP s, NEG	↕	↕	V	↕	1	↕	8-bit subtract, subtract with carry, compare and negate accumulator
AND s	0	↕	P	↕	0	1	Logical operations
OR s; XOR s	0	↕	P	↕	0	0	And set's different flags
INC s	•	↕	V	↕	0	↕	8-bit increment
DEC m	•	↕	V	↕	1	↕	8-bit decrement
ADD DD, ss	↕	•	•	•	0	X	16-bit add
ADC HL, ss	↕	↕	V	↕	0	X	16-bit add with carry
SBC HL, ss	↕	↕	V	↕	1	X	16-bit subtract with carry
RLA; RLCA, RRA, RRCA	↕	•	•	•	0	0	Rotate accumulator
RL m; RLC m; RR m; RRC m SLA m; SRA m; SRL m	↕	↕	P	↕	0	0	Rotate and shift location m
RLD, RRD	•	↕	P	↕	0	0	Rotate digit left and right
DAA	↕	↕	P	↕	•	↕	Decimal adjust accumulator
CPL	•	•	•	•	1	1	Complement accumulator
SCF	1	•	•	•	0	0	Set carry
CCF	↕	•	•	•	0	X	Complement carry
IN r, (C)	•	↕	P	↕	0	0	Input register indirect
INI; IND; OUTI; OUTD	•	↕	X	X	1	X	Block input and output
INIR; INDR; OTIR; OTDR	•	1	X	X	1	X	Z = 0 if B ≠ 0 otherwise Z = 1
LDI, LDD	•	X	↕	X	0	0	Block transfer instructions
LDIR, LDDR	•	X	0	X	0	0	P/V = 1 if BC ≠ 0, otherwise P/V = 0
CPI, CPIR, CPD, CPDR	•	↕	↕	↕	1	X	Block search instructions Z = 1 if A = (HL), otherwise Z = 0 P/V = 1 if BC ≠ 0, otherwise P/V = 0
LD A, I; LD A, R	•	↕	IFF	↕	0	0	The content of the interrupt enable flip-flop (IFF) is copied into the P/V flag
BIT b, s	•	↕	X	X	0	1	The state of bit b of location s is copied into the Z flag
NEG	↕	↕	V	↕	1	↕	Negate accumulator

The following notation is used in this table:

Symbol	Operation
C	Carry/link flag. C=1 if the operation produced a carry from the MSB of the operand or result.
Z	Zero flag. Z=1 if the result of the operation is zero.
S	Sign flag. S=1 if the MSB of the result is one.
P/V	Parity or overflow flag. Parity (P) and overflow (V) share the same flag. Logical operations affect this flag with the parity of the result while arithmetic operations affect this flag with the overflow of the result. If P/V holds parity, P/V=1 if the result of the operation is even, P/V=0 if result is odd. If P/V holds overflow, P/V=1 if the result of the operation produced an overflow.
H	Half-carry flag. H=1 if the add or subtract operation produced a carry into or borrow from into bit 4 of the accumulator.
N	Add/Subtract flag. N=1 if the previous operation was a subtract.
	H and N flags are used in conjunction with the decimal adjust instruction (DAA) to properly correct the result into packed BCD format following addition or subtraction using operands with packed BCD format.
↕	The flag is affected according to the result of the operation.
•	The flag is unchanged by the operation.
0	The flag is reset by the operation.
1	The flag is set by the operation.
X	The flag is a "don't care."
V	P/V flag affected according to the overflow result of the operation.
P	P/V flag affected according to the parity result of the operation.
r	Any one of the CPU registers A, B, C, D, E, H, L.
s	Any 8-bit location for all the addressing modes allowed for the particular instruction. .
ss	Any 16-bit location for all the addressing modes allowed for that instruction.
ii	Any one of the two index registers IX or IY.
R	Refresh counter.
n	8-bit value in range <0, 255>
nn	16-bit value in range <0, 65535>
m	Any 8-bit location for all the addressing modes allowed for the particular instruction.

Vanaf de allereerste beginselen van de machinetaal tot aan de werking van 's werelds meest bekende micro-processor, de Z80a: met behulp van dit boek zult u in relatief korte tijd in staat zijn zelf machinetaal-programma's voor de MSX (Z80) computers te schrijven.
In dit boek is een volledige beschrijving van de systeemroutines opgenomen, met waar nodig listings die het geheel nader verklaren.
Kortom: een gefungeerde eerste stap in het programmeren met de enige taal die een computer werkelijk begrijpt.

ISBN 90 229 3360 1

MSX Bibliotheek 4

DATA BECKER
NEDERLANDS*